Constructing Automata from Temporal Logic Formulas : A Tutorial*

Pierre Wolper

Université de Liège,
Institut Montefiore, B28,
4000 Liège, Belgium
pw@montefiore.ulg.ac.be,
http://www.montefiore.ulg.ac.be/~pw/

Abstract. This paper presents a tutorial introduction to the construction of finite-automata on infinite words from linear-time temporal logic formulas. After defining the source and target formalisms, it describes a first construction whose correctness is quite direct to establish, but whose behavior is always equal to the worst-case upper bound. It then turns to the techniques that can be used to improve this algorithm in order to obtain the quite effective algorithms that are now in use.

1 Introduction

Model checking [CES86,QS81,VW86] is a widespread technique for verifying temporal properties of reactive programs. There are several ways to develop the theory of model checking, a particularly attractive one being through the construction of automata from temporal logic formulas [VW86,BVW94]. As a result, there has been a fair amount of interest in the construction of automata from temporal logical formulas, the history of which is actually fairly interesting.

The starting point is clearly the work of Büchi on the decidability of the first and second-order monadic theories of one successor [Büc62]. These decidability results were obtained through a translation to infinite-word automata, for which Büchi had to prove a very nontrivial complementation lemma. The translation is nonelementary, but this is the best that can be done. It is quite obvious that linear-time temporal logic can be translated to the first-order theory of one successor and hence to infinite-word automata. From a logician's point of view, this could be seen as settling the question, but an interest in using temporal logic for computer science applications, in particular program synthesis [MW84,EC82] triggered a second look at the problem. Indeed, it was rather obvious that a nonelementary construction was not necessary to build an automaton from a temporal logic formula; it could be done within a single exponential by a direct construction [WVS83,VW94]. As originally presented, this

* This work was partially funded by a grant of the "Communauté française de Belgique - Direction de la recherche scientifique - Actions de recherche concertées".

construction was worst and best case exponential. Though it was fairly clear that it could be modified to operate more effectively on many instances, nothing was written about this, probably because the topic was thought to be rather trivial and had no bearing on general complexity results.

Nevertheless, the idea of doing model checking through the construction of automata was taken seriously, at least by some, and attempts were made to incorporate automata-theoretic model checking into tools, notably into SPIN [Hol91,Hol97]. Of course, this required an effective implementation of the logic to automaton translation algorithm and the pragmatics of doing this are not entirely obvious. A description of such an implementation was given in [GPVW95] and improved algorithms have been proposed since [DGV99,SB00]. Note that there are some questions about how to measure such improvements since the worst-case complexity of the algorithms stays the same. Nevertheless, experiments show that, for the temporal logic formulas most frequently used in verification, the automata can be kept quite small. Thus, even though it is an intrinsically exponential process, building an automaton from a temporal logic formula appears to be perfectly feasible in practice. What is surprising is that it took quite a long time for the details of a usable algorithmic solution to be developed and codified.

The goal of this paper is to provide a tutorial introduction to the construction of Büchi infinite-word automata from linear temporal logic formulas. After an introduction to temporal logic and a presentation of infinite-word automata that stresses their kinship to logic, a first simple, but always exponential, construction is presented. This construction is similar to the one of [WVS83,VW94], but is more streamlined since it does not deal with the extended temporal logic considered in the earlier work. Thereafter, it is shown how this construction can be adapted to obtain a more effective construction that only builds the needed states of the automaton, as described in [GPVW95] and further improved in [DGV99,SB00].

2 Linear-Time Temporal Logic

Linear-time temporal logic is an extension of propositional logic geared to reasoning about infinite sequences of states. The sequences considered are isomorphic to the natural numbers and each state is a propositional interpretation. The formulas of the logic are built from atomic propositions using Boolean connectives and temporal operators. Purely propositional formulas are interpreted in a single state and the temporal operators indicate in which states of a sequence their arguments must be evaluated.

Formally, the formulas of linear-time temporal logic (LTL) built from a set of atomic propositions P are the following:

- **true**, **false**, p, and $\neg p$, for all $p \in P$;
- $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are LTL formulas;
- $\bigcirc \varphi_1$, $\varphi_1 \, U \, \varphi_2$, and $\varphi_1 \, \tilde{U} \, \varphi_2$, where φ_1 and φ_2 are LTL formulas.

The operator $\bigcirc$ is read "next" and means in the *next* state. The operator U is read "until" and requires that its first argument be true *until* its second argument is true, which is required to happen. The operator $\tilde{U}$ is the dual of U and is best read as "releases", since it requires that its second argument always be true, a requirement that is *released* as soon as its first argument becomes true. Two derived operators are in very common use. They are

– $\Diamond \varphi = \mathbf{true}\, U\, \varphi$, which is read "eventually" and requires that its argument be true *eventually*, i.e. at some point in the future; and
– $\Box \varphi = \mathbf{false}\, \tilde{U}\, \varphi$, which is read "always" and requires that its argument be true *always*, i.e. at all future points.

Formally, the semantics of LTL is defined with respect to sequences $\sigma : \mathbf{N} \to 2^P$. For a sequence σ, σ^i represents the suffix of σ obtained by removing its i first states, i.e. $\sigma^i(j) = \sigma(i + j)$. The truth value of a formula on a sequence σ, which is taken to be the truth value obtained by starting the interpretation of the formula in the first state of the sequence, is given by the following rules:

– For all σ, we have $\sigma \models \mathbf{true}$ and $\sigma \not\models \mathbf{false}$;
– $\sigma \models p$ for $p \in P$ iff $p \in \sigma(0)$;
– $\sigma \models \neg p$ for $p \in P$ iff $p \notin \sigma(0)$;
– $\sigma \models \varphi_1 \wedge \varphi_2$ iff $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$;
– $\sigma \models \varphi_1 \vee \varphi_2$ iff $\sigma \models \varphi_1$ or $\sigma \models \varphi_2$;
– $\sigma \models \bigcirc \varphi_1$ iff $\sigma^1 \models \varphi_1$;
– $\sigma \models \varphi_1\, U\, \varphi_2$ iff there exists $i \geq 0$ such that $\sigma^i \models \varphi_2$ and for all $0 \leq j < i$, we have $\sigma^j \models \varphi_1$;
– $\sigma \models \varphi_1\, \tilde{U}\, \varphi_2$ iff for all $i \geq 0$ such that $\sigma^i \not\models \varphi_2$, there exists $0 \leq j < i$ such that $\sigma^j \models \varphi_1$.

In the logic we have defined, negation is only applied to atomic propositions. This restriction can be lifted with the help of the following relations, which are direct consequences of the semantics we have just given:

$$\sigma \not\models \varphi_1\, U\, \varphi_2 \text{ iff } \sigma \models (\neg\varphi_1)\, \tilde{U} (\neg\varphi_2)$$

$$\sigma \not\models \varphi_1\, \tilde{U}\, \varphi_2 \text{ iff } \sigma \models (\neg\varphi_1)\, U (\neg\varphi_2)$$

$$\sigma \not\models \bigcirc \varphi_1 \text{ iff } \sigma \models \bigcirc \neg\varphi_1.$$

To easily understand the link between temporal logic formulas and automata, it is useful to think of a temporal formula as being a description of a set of infinite sequences: those that satisfy it. Note that to check that a sequence satisfies a temporal logic formula φ, a rather natural way to proceed is to attempt to label each state of the sequence with the subformulas of φ that are true there. One would proceed outwards, starting with the propositional subformulas, and adding exactly those subformulas that are compatible with the semantic rules. Of course, for an infinite sequence, this cannot be done effectively. However, this abstract procedure will turn out to be conceptually very useful.

3 Automata on Infinite Words

Infinite words (or ω-words) are sequences of symbols isomorphic to the natural numbers. Precisely, an infinite word over an alphabet Σ is a mapping $w : \mathbf{N} \to \Sigma$.

An automaton on infinite words is a structure that defines a set of infinite words. Even though infinite word automata look just like traditional automata, one gets a better understanding of them by not considering them as operational objects but, rather, by seeing them as descriptions of sets of infinite sequences, and hence as a particular type of logical formula.

We will consider Büchi and generalized Büchi automata on infinite words. A Büchi infinite word automaton has exactly the same structure as a traditional finite word automaton. It is a tuple $A = \{\Sigma, S, \delta, S_0, F\}$ where

- Σ is an alphabet,
- S is a set of states,
- $\delta : S \times \Sigma \to S$ (deterministic) or $\delta : S \times \Sigma \to 2^S$ (nondeterministic) is a transition function,
- $S_0 \subseteq S$ is a set of initial states (a singleton for deterministic automata), and
- $F \subseteq S$ is a set of accepting states.

What distinguishes a Büchi infinite-word automaton from a finite word automaton is that its semantics are defined over infinite words. Let us now examine these semantics using a somewhat logical point of view. A word w is accepted by an automaton $A = \{\Sigma, S, \delta, S_0, F\}$ (the word satisfies the automaton) if there is a labeling

$$\rho : \mathbf{N} \to S$$

of the word by states such that

- $\rho(0) \in S_0$ (the initial label is an initial state),
- $\forall 0 \leq i, \rho(i+1) \in \delta(\rho(i), w(i))$ (the labeling is compatible with the transition relation),
- $\inf(\rho) \cap F \neq \emptyset$ where $\inf(\rho)$ is the set of states that appear infinitely often in ρ (the set of repeating states intersects F).

Example 1. The automaton of Figure 1 accepts all words over the alphabet $\Sigma = \{a, b\}$ that contain b infinitely often.

Generalized Büchi automata differ from Büchi automata by their acceptance condition. The acceptance condition of a generalized Büchi automaton is a set of sets of states $\mathcal{F} \subseteq 2^S$, and the requirement is that some state of each of the sets $F_i \in \mathcal{F}$ appears infinitely often. More formally, a generalized Büchi $A = \{\Sigma, S, \delta, S_0, \mathcal{F}\}$ accepts a word w if there is a labeling ρ of w by states of A that satisfies the same first two conditions as given for Büchi automata, the third being replaced by:

- For each $F_i \in \mathcal{F}$, $\inf(\rho) \cap F_i \neq \emptyset$.

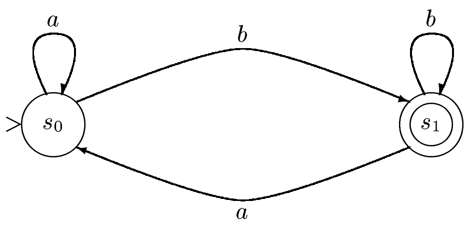

Fig. 1. An automaton accepting all words over $\Sigma = \{a, b\}$ containing b infinitely often

As the following lemma shows, generalized Büchi automata accept exactly the same languages as Büchi automata.

Lemma 1. *Given a generalized Büchi automaton, one can construct an equivalent Büchi automaton.*

Proof. Given a generalized Büchi automaton $A = (\Sigma, S, \delta, S_0, \mathcal{F})$, where $\mathcal{F} = \{F_1, \ldots, F_k\}$, the Büchi automaton $A' = (\Sigma, S', \delta', S'_0, F')$ defined as follows accepts the same language as A.

- $S' = S \times \{1, \ldots, k\}$.
- $S'_0 = S_0 \times \{1\}$.
- δ' is defined by $(t, i) \in \delta'((s, j), a)$ if

$$t \in \delta(s, a) \text{ and } \begin{cases} i = j & \text{if } s \notin F_j, \\ i = (j \bmod k) + 1 & \text{if } s \in F_j. \end{cases}$$

- $F' = F_1 \times \{1\}$.

The idea of the construction is that the states of A' are the states of A marked by an integer in the range $[1, k]$. The mark is unchanged unless one goes through a state in F_j, where j is the current value of the mark. In that case the mark is incremented (reset to 1 if it is k). If one repeatedly cycles through all the marks, which is necessary for F' to be reached infinitely often, then all sets in $\mathcal{F}$ are visited infinitely often. Conversely, if it is possible to visit all sets in $\mathcal{F}$ infinitely often in A, it is possible to do so in the order F_1, F_2, $\ldots F_k$ and hence to infinitely often go through F' in A'.

Example 2. Figure 3 shows the Büchi automaton equivalent to the generalized Büchi automaton of Figure 2 whose acceptance condition is $\mathcal{F} = \{\{s_0\}, \{s_1\}\}$.

Nondeterministic[1] Büchi automata have many interesting properties. In particular they are closed under all Boolean operations as well as under projection. Closure under union, projection are immediate given that we are dealing

[1] Deterministic Büchi automata are less powerful and do not enjoy the same properties (see for instance [Tho90]).

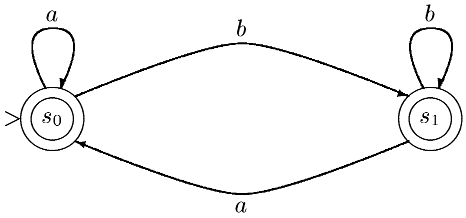

Fig. 2. A generalized Büchi automaton

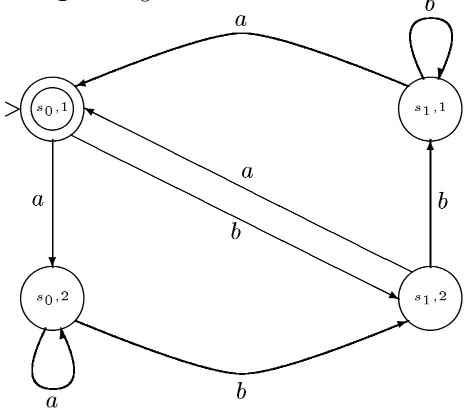

Fig. 3. From generalized Büchi to Büchi

with nondeterministic automata; closure under intersection is obtained using a product construction similar to the one employed for finite-word automata. Closure under complementation is much more tricky and has been the subject of an extensive literature [Büc62,SVW87,Saf88,KV97]. Checking that a (generalized) Büchi automaton is nonempty (accepts at least one word) can be done by computing its strongly connected components, and checking that there exists a reachable strongly connected component that has a non empty intersection with each set in $\mathcal{F}$.

4 From Temporal Logic to Automata

4.1 Problem Statement

We now consider the following problem: given an LTL formula φ built from a set of atomic propositions P, construct an automaton on infinite words over the alphabet 2^P that accepts exactly the infinite sequences satisfying φ.

To get an intuitive idea of what we are aiming at, let us first look at an example.

Example 3. Consider the formula $\Diamond p$. This formula describes the sequences over $\{\emptyset, \{p\}\}$ in which $\{p\}$ occurs at least once. These sequences are accepted by the automaton of Figure 4.

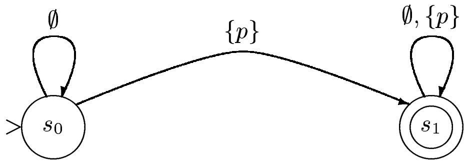

Fig. 4. An automaton for $\Diamond p$

4.2 The Closure of a Formula

In order to develop a procedure for building automata from LTL formulas, we first look at the problem of determining if a sequence $\sigma : \mathbf{N} \rightarrow 2^P$ satisfies a formula φ defined over the set of propositions P. This can, at least conceptually, be done by labeling the sequence with subformulas of φ in a way that respects LTL semantics. First, let us define the set of subformulas of a formula φ that are needed. This set is called the *closure* of φ ($cl(\varphi)$) and is defined as follows:

- $\varphi \in cl(\varphi)$,
- $\varphi_1 \wedge \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$,
- $\varphi_1 \vee \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$,
- $\bigcirc \varphi_1 \in cl(\varphi) \Rightarrow \varphi_1 \in cl(\varphi)$,
- $\varphi_1 \, U \, \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$,
- $\varphi_1 \, \tilde{U} \, \varphi_2 \in cl(\varphi) \Rightarrow \varphi_1, \varphi_2 \in cl(\varphi)$.

Example 4.

$$cl(\Diamond \neg p) = cl(\mathbf{true} \, U \, \neg p) = \{\Diamond \neg p, \neg p, \mathbf{true}\}$$

4.3 Rules for Labeling Sequences

The next step is to define the set of rules that a valid *closure labeling* $\tau : \mathbf{N} \rightarrow 2^{cl(\varphi)}$ of a sequence $\sigma : \mathbf{N} \rightarrow 2^P$ has to satisfy. The validity criterion is that, if a formula $\varphi_1 \in cl(\varphi)$ labels a position i (i.e. $\varphi_1 \in \tau(i)$), then the sequence σ^i satisfies it ($\sigma^i \models \varphi_1$)[2]. For this to hold, our labeling rules have to mirror the semantic rules for LTL. A first set of rules deals with the purely propositional part of LTL.

Consider a closure labeling $\tau : \mathbf{N} \rightarrow 2^{cl(\varphi)}$ of a sequence $\sigma : \mathbf{N} \rightarrow 2^P$ for a formula φ defined over a set of atomic propositions P. For τ to be a valid labeling, it has to satisfy the following rules for every $i \geq 0$:

1. $\mathbf{false} \notin \tau(i)$;
2. for $p \in P$, if $p \in \tau(i)$ then $p \in \sigma(i)$, and if $\neg p \in \tau(i)$ then $p \notin \sigma(i)$;
3. if $\varphi_1 \wedge \varphi_2 \in \tau(i)$ then $\varphi_1 \in \tau(i)$ and $\varphi_2 \in \tau(i)$;

[2] Such a validly labeled structure is often called a Hintikka structure in the modal logic literature

4. if $\varphi_1 \vee \varphi_2 \in \tau(i)$ then $\varphi_1 \in \tau(i)$ or $\varphi_2 \in \tau(i)$.

Note that the labeling rules are "if" rules and not "if and only if" rules. They give the requirements that a valid closure labeling *must* satisfy, but they do not require that labelings be maximal: there can be formulas of the closure that are satisfied at a given position, but that are not included in the label of that position.

Let us now turn to elements of the closure whose main operator is temporal. For the operator $\bigcirc$, the rule is quite immediate. We have that for all $i \geq 0$,

5. if $\bigcirc \varphi_1 \in \tau(i)$ then $\varphi_1 \in \tau(i+1)$.

For the U and $\tilde{U}$ operators, the semantic rules refer to a possibly infinite set of points of the sequence, which we would like to avoid in our labeling rules. Fortunately, this is mostly possible. Indeed, one can fairly easily show from the semantic rules that the following identities hold:

$$\begin{aligned} \varphi_1 \, U \, \varphi_2 &\equiv (\varphi_2 \vee (\varphi_1 \wedge \bigcirc(\varphi_1 \, U \, \varphi_2))) \\ \varphi_1 \, \tilde{U} \, \varphi_2 &\equiv (\varphi_2 \wedge (\varphi_1 \vee \bigcirc(\varphi_1 \, \tilde{U} \, \varphi_2))). \end{aligned}$$

These identities then suggest the following labeling rules for all positions $i \geq 0$:

6. if $\varphi_1 \, U \, \varphi_2 \in \tau(i)$ then either $\varphi_2 \in \tau(i)$, or $\varphi_1 \in \tau(i)$ and $\varphi_1 \, U \, \varphi_2 \in \tau(i+1)$;
7. if $\varphi_1 \, \tilde{U} \, \varphi_2 \in \tau(i)$ then $\varphi_2 \in \tau(i)$, and either $\varphi_1 \in \tau(i)$ or $\varphi_1 \, \tilde{U} \, \varphi_2 \in \tau(i+1)$.

The rule for $\tilde{U}$ is sufficient to ensure that the labeling is valid. Unfortunately, the same is not true for the operator U. Indeed, rule 6 does not force the existence of a point at which φ_2 appears: such a point can be postponed forever. We thus need to add one more labeling rule, which unfortunately does not only refer to consecutive points. For every position $i \geq 0$, we must have that

8. if $\varphi_1 \, U \, \varphi_2 \in \tau(i)$ then there is a $j \geq i$ such that $\varphi_2 \in \tau(j)$.

As a hint at the requirement expressed by rule 8, a formula of the form $\varphi_1 \, U \, \varphi_2$ is often referred to as an *eventuality* since it requires that the formula φ_2 be *eventually* true. Rule 8 is then said to require that the eventualities are *fulfilled*.

We can now formalize the fact that the labeling rules we have given characterize the valid labelings. First we show that the labeling rules only allow valid labelings.

Lemma 2. *Consider a formula φ defined over a set of propositions P, a sequence $\sigma : \mathbf{N} \rightarrow 2^P$, and a closure labeling $\tau : \mathbf{N} \rightarrow 2^{cl(\varphi)}$ satisfying rules 1–8. For every formula $\varphi' \in cl(\varphi)$ and $i \geq 0$, one has that if $\varphi' \in \tau(i)$ then $\sigma^i \models \varphi'$.*

Proof. The proof proceeds by structural induction on the formulas of $cl(\varphi)$. Let us consider the most interesting case, which is that of a formula φ' of the form $\varphi_1 \, U \, \varphi_2$. By rule 8, one has that there is a $j \geq i$ such that $\varphi_2 \in \tau(j)$ and, by inductive hypothesis, such that $\sigma^j \models \varphi_2$. Consider the smallest such j and a k such that $i \leq k < j$. Since $\varphi_1 \, U \, \varphi_2 \in \tau(i)$, and since for all $i \leq k' \leq k$, $\varphi_2 \notin \tau(k')$, rule 6 implies that $\varphi_1 \, U \, \varphi_2 \in \tau(k)$ and also that $\varphi_1 \in \tau(k)$. Hence, by inductive hypothesis, $\sigma^k \models \varphi_1$.

Next we need to establish that when a sequence satisfies a formula, a closure labeling satisfying rules 1–8 exists.

Lemma 3. *Consider a formula φ defined over a set of propositions P and a sequence $\sigma : \mathbf{N} \to 2^P$. If $\sigma \models \varphi$, there exists a closure labeling $\tau : \mathbf{N} \to 2^{cl(\varphi)}$ satisfying rules 1–8 and such that $\varphi \in \tau(0)$.*

Proof. Consider the closure labeling defined by $\varphi' \in \tau(i)$ iff $\sigma^i \models \varphi'$ for all $\varphi' \in cl(\varphi)$. Given that $\sigma \models \varphi$, one immediately has that $\varphi \in \tau(0)$. Furthermore, that fact that rules 1–8 are satisfied is a direct consequence of the semantics of LTL.

The following theorem is then a direct consequence of Lemmas 2 and 3.

Theorem 1. *Consider a formula φ defined over a set of propositions P and a sequence $\sigma : \mathbf{N} \to 2^P$. One then has that $\sigma \models \varphi$ iff there is a closure labeling $\tau : \mathbf{N} \to 2^{cl(\varphi)}$ of σ satisfying rules 1–8 and such that $\varphi \in \tau(0)$.*

4.4 Defining the Automaton

Given Theorem 1, the construction of an automaton accepting the sequences satisfying a formula φ is almost immediate. Indeed, remember that an automaton accepts an ω-sequence when this sequence can be labeled by states of the automaton, while satisfying the constraints imposed by the transition relation as well as by the initial and accepting state sets. The idea is simply to use $2^{cl(\varphi)}$ as state set and hence as set of possible labels. It then remains to express the required properties of the labeling by an appropriate definition of the structure of the automaton. We now show how this can be done.

Given a formula φ, a generalized Büchi automaton accepting exactly the sequences $\sigma : \mathbf{N} \to 2^P$ satisfying φ can be defined as follows. The automaton is $A_\varphi = (\Sigma, S, \delta, S_0, \mathcal{F})$ where

- $\Sigma = 2^P$,
- The set of states S is the set of possible labels, i.e. the subsets $\mathbf{s}$ of $2^{cl(\varphi)}$ that satisfy
  - **false** $\notin \mathbf{s}$;
  - if $\varphi_1 \wedge \varphi_2 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{s}$ and $\varphi_2 \in \mathbf{s}$;
  - if $\varphi_1 \vee \varphi_2 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{s}$ or $\varphi_2 \in \mathbf{s}$.

  The states (and hence possible labels) are thus the subsets of $2^{cl(\varphi)}$ that satisfy rules 1 as well as rules 3 and 4.
- The transition function δ checks that the propositional labeling matches the one in the sequence being considered (rule 2) and that the rules 5–7 for the temporal operators are satisfied. Thus, $\mathbf{t} \in \delta(\mathbf{s}, \mathbf{a})$ iff

- For all $p \in P$, if $p \in \mathbf{s}$ then $p \in \mathbf{a}$.
- For all $p \in P$, if $\neg p \in \mathbf{s}$ then $p \notin \mathbf{a}$.
- if $\bigcirc \varphi_1 \in \mathbf{s}$ then $\varphi_1 \in \mathbf{t}$.
- if $\varphi_1 \, U \, \varphi_2 \in \mathbf{s}$ then either $\varphi_2 \in \mathbf{s}$, or $\varphi_1 \in \mathbf{s}$ and $\varphi_1 \, U \, \varphi_2 \in \mathbf{t}$.
- if $\varphi_1 \, \tilde{U} \, \varphi_2 \in \mathbf{s}$ then $\varphi_2 \in \mathbf{s}$ and either $\varphi_1 \in \mathbf{s}$, or $\varphi_1 \, \tilde{U} \, \varphi_2 \in \mathbf{t}$.

– The set of initial states is defined in order to ensure that φ appears in the label of the first position of the sequence. We thus have that $S_0 = \{\mathbf{s} \in S \mid \varphi \in \mathbf{s}\}$.
– The acceptance condition $\mathcal{F}$ is used to impose rule 8 on the fulfillment of eventualities, but seeing how this can be done requires a slightly closer look at this requirement.

What needs to be imposed to satisfy rule 8 is that, for every eventuality formula $\varphi_1 \, U \, \varphi' \equiv e(\varphi') \in cl(\varphi)$, any state that contains that formula is followed by a state that contains φ'. The problem with the way this requirement is stated is that it requires "remembering" that a state in which $e(\varphi')$ occurs has been seen and hence extending the set of states of the automaton. Fortunately, this can be avoided.

Rule 6 (and hence the transition relation of the automaton) requires that if an eventuality $e(\varphi')$ appears, it keeps on appearing until the first state in which φ' appears. So, the only problematic situation would be one in which $e(\varphi')$ appears indefinitely without φ' ever appearing. So its is sufficient to require that the automaton goes infinitely often through a state in which both $e(\varphi')$ and φ' appear or in which $e(\varphi')$ does not appear, the latter case allowing for the eventuality no longer to be required after some point. The acceptance condition of the automaton is thus the following generalized Büchi condition.

– If the eventualities appearing in $cl(\varphi)$ are $e_1(\varphi_1), \ldots e_m(\varphi_m)$, $\mathcal{F} = \{\Phi_1, \ldots, \Phi_m\}$ where $\Phi_i = \{\mathbf{s} \in S \mid e_i, \varphi_i \in \mathbf{s} \vee e_i \notin \mathbf{s}\}$.

Given the way we have expressed the semantics of LTL in terms of labeling rules and given the semantics of Büchi automata, the correctness of the construction we have just given is essentially immediate.

Example 5. The automaton for $\Diamond p$ is given in Figure 5, where $\mathcal{F} = \{\{1, 3, 4\}\}$.

Note that in this example we have already applied two optimizations to the construction. First, we have identified states containing **true** with the states defined by an otherwise identical set of formulas. Second, we have omitted the transitions leaving from nodes 3 and 4 to nodes 1 and 2. It is intuitively obvious that these transitions are not needed but, in the next section, we will generalize these types of optimizations and justify them precisely.

5 Improving the Construction

5.1 Omitting Redundant Transitions

The states of the automaton we build for a formula are subsets of the closure of that formula. The subset ordering thus naturally defines a partial order on the

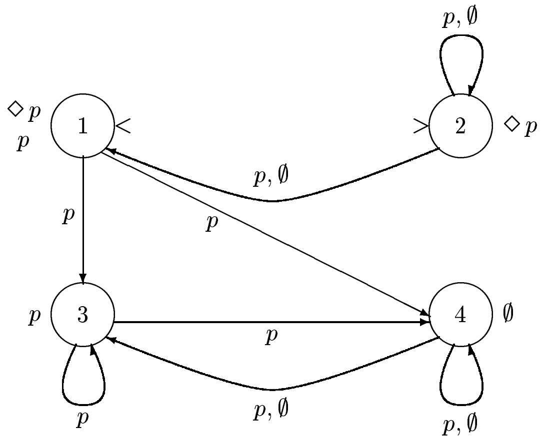

Fig. 5. The automaton constructed for $\diamond p$

automaton states. Furthermore, given the way the transition relation is defined, any transition possible from a state $\mathbf{s}_1$ is also possible from any state $\mathbf{s}_2 \subset \mathbf{s}_1$. This seems to imply that if, from a given state, two transitions lead to states $\mathbf{s}_1$ and $\mathbf{s}_2$ such that $\mathbf{s}_2 \subset \mathbf{s}_1$, then it is sufficient to keep the transition leading to the state $\mathbf{s}_2$. Almost so. Indeed, the way the transition relation of the automaton is defined guarantees that if there is a computation of the automaton on a given word from $\mathbf{s}_1$, there is also one from $\mathbf{s}_2$. The problem is with accepting states: if $\mathbf{s}_1$ contains an eventuality formula $e(\varphi')$ as well as its argument φ', but that $\mathbf{s}_2$ only contains $e(\varphi')$, $\mathbf{s}_2$ might be outside an accepting set in which $\mathbf{s}_1$ is included. The simplification rule we will use is thus the following.

Omit transitions. Assume that from a state $\mathbf{s}$ two identically labeled transitions lead to states $\mathbf{s}_1$ and $\mathbf{s}_2$ such that $\mathbf{s}_2 \subset \mathbf{s}_1$ and such that, for all eventuality formulas $e(\varphi') \in \mathbf{s}_1$, if $e(\varphi') \in \mathbf{s}_2$ and $\varphi' \in \mathbf{s}_1$ then also $\varphi' \in \mathbf{s}_2$. The transition from $\mathbf{s}$ to $\mathbf{s}_1$ can then be omitted.

Example 6. Applying the **omit transitions** rule to the automaton of Figure 5 and eliminating unreachable states, one obtains the automaton of Figure 6.

To see that the **omit transitions** rule is sound, we establish that for every state of the automaton, the language accepted from that state after applying the **omit transitions** rule is unchanged. First notice that we are removing transitions. So, after applying the rule, the language accepted cannot contain more words. We show that it also cannot contain less words. Assume that there exists an accepting computation from a state $\mathbf{s}$ before applying the **omit transitions** rule. Such a computation still exists after applying the rule. Indeed, if the computation from $\mathbf{s}$ starts with an omitted transition leading to a state $\mathbf{s}_1$, there remains an identically labeled transition to a state $\mathbf{s}_2 \subset \mathbf{s}_1$ that is accepting whenever $\mathbf{s}_1$ is accepting. Now, since $\mathbf{s}_2 \subset \mathbf{s}_1$, before the transition omission procedure, all transitions possible from $\mathbf{s}_1$ are also possible from $\mathbf{s}_2$, so there also is an accepting computation from $\mathbf{s}_2$. Of course, some transitions from $\mathbf{s}_2$, may also

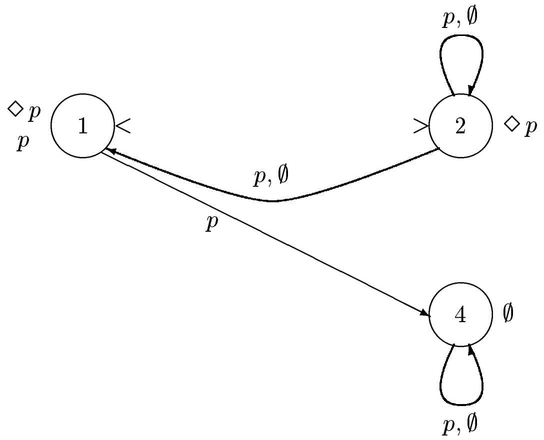

Fig. 6. A simplified automaton for $\Diamond p$

have been omitted, but the same argument can be repeated for the computation starting at $\mathbf{s}_2$. By induction, one can then conclude the existence of the required accepting computation in the simplified automaton.

5.2 Building the Automaton by Need

The most obviously wasteful aspect of the construction we have shown is that it defines the set of states to be all subsets of the closure that satisfy the rules 1, 3, and 4. Indeed, many of these states might not be reachable from initial states, especially if the **omit transitions** simplification rule is applied. To avoid this, we are going to construct the states of the automaton as needed, starting with the initial states and adding the states that must appear as targets of transitions.

Preparing to do this, notice that all the rules embodied in the transitions of the automaton require that, if some formula of the closure occurs in the current state, then some other formula also occurs in the current or next state. Furthermore, the **omit transitions** simplification allows us to only consider the minimal states satisfying these conditions. This leads us to defining an operation that adds to a subset of the closure all formulas that *must* be true in the current and in the immediately next state. For ease of definition, we define this operation (*saturate*) not on subsets of the closure, but on sets of subsets of the closure.

Let $\mathbf{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k\} \subseteq 2^{cl(\varphi)}$, then we define $\mathit{saturate}(\mathbf{Q})$ as follows.

1. Repeat until stabilization: for each $\mathbf{q}_i \in \mathbf{Q}$,
   (a) If $\varphi_1 \wedge \varphi_2 \in \mathbf{q}_i$, then
   $\mathbf{Q} := \mathbf{Q} \setminus \{\mathbf{q}_i\} \cup \{\mathbf{q}_i \cup \{\varphi_1, \varphi_2\}\}$;
   (b) If $\varphi_1 \vee \varphi_2 \in \mathbf{q}_i$, then
   $\mathbf{Q} := \mathbf{Q} \setminus \{\mathbf{q}_i\} \cup \{\mathbf{q}_i \cup \{\varphi_1\}\} \cup \{\mathbf{q}_i \cup \{\varphi_2\}\}$;
   (c) If $\varphi_1 \, U \, \varphi_2 \in \mathbf{q}_i$, then
   $\mathbf{Q} := \mathbf{Q} \setminus \{\mathbf{q}_i\} \cup \{\mathbf{q}_i \cup \{\varphi_2\}\} \cup \{\mathbf{q}_i \cup \{\varphi_1, \bigcirc(\varphi_1 \, U \, \varphi_2)\}\}$;

(d) If $\varphi_1 \tilde{U} \varphi_2 \in \mathbf{q}_i$, then
$\mathbf{Q} := \mathbf{Q} \setminus \{\mathbf{q}_i\} \cup \{\mathbf{q}_i \cup \{\varphi_1, \varphi_2\}\} \cup \{\mathbf{q}_i \cup \{\varphi_2, \bigcirc(\varphi_1 \tilde{U} \varphi_2)\}\}$

2. Remove all $\mathbf{q}_i \in \mathbf{Q}$ such that $\mathbf{false} \in \mathbf{q}_i$

If the operation *saturate* is applied to a singleton $\mathbf{q}$, then the result is a set of sets of formulas[3] that represent possible ways of satisfying the requirements expressed by the formulas in $\mathbf{q}$. Among such sets of formulas, we will be especially interested in the propositional formulas and in the formulas having $\bigcirc$ as their main connective, which constrain the next state. We thus define the following filters on a set of LTL formulas $\mathbf{q}$:

1. $X(\mathbf{q}) = \{\varphi_i \mid \bigcirc\varphi_i \in \mathbf{q}\}$ (the "next" formulas in $\mathbf{q}$ with their $\bigcirc$ operator stripped),
2. $P(\mathbf{q}) = \{p_i \mid p_i \in \mathbf{q} \wedge p_i \in P\}$ (the atomic propositions in $\mathbf{q}$),
3. $nP(\mathbf{q}) = \{p_i \mid \neg p_i \in \mathbf{q} \wedge p_i \in P\}$ (the negated atomic propositions in $\mathbf{q}$).

We are now ready to give a "by need" algorithm for generating the automaton. For a formula φ, the algorithm generates an automaton $A_\varphi = (\Sigma, S, \delta, S_0, \mathcal{F})$. The alphabet and accepting condition are defined as before. The states and transitions are progressively generated by the algorithm. For ease of notation, we will represent the transition function δ as a set of triples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ where $\mathbf{s}, \mathbf{s}' \in S$ and $\mathbf{a} \in \Sigma$.

The algorithm works with a list of unprocessed states for which successors still have to be generated. For these unprocessed states, additional information in the form of "next requirements", i.e. the formulas that have to be true in all the immediate successors of the unprocessed state, is maintained. Unprocessed states thus take the form of a pair of sets of formulas $(\mathbf{s}, \mathbf{x})$, where $\mathbf{s}$ is the state and $\mathbf{x}$ the "next requirements". The unprocessed states are stored in a list *unp*. The automaton building procedure is then the following.

build-auto (φ)
1. $S := \emptyset$; $\delta := \emptyset$; $S_0 := \{\mathbf{q} \cap cl(\varphi) \mid \mathbf{q} \in saturate(\{\{\varphi\}\})\}$
2. $unp := \{(\mathbf{q} \cap cl(\varphi), X(\mathbf{q})) \mid \mathbf{q} \in saturate(\{\{\varphi\}\})\}$
3. **while** $unp \neq \emptyset$ **do**
   Choose and remove $(\mathbf{s}, \mathbf{x})$ from *unp*;
   $S := S \cup \{\mathbf{s}\}$;
   For each $\mathbf{q} \in saturate(\{\mathbf{x}\})$ **do**
      For each $\mathbf{a} \in \Sigma$ **such that** $P(\mathbf{s}) \subseteq \mathbf{a} \wedge nP(\mathbf{s}) \cap \mathbf{a} = \emptyset$
         $\delta := \delta \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{q} \cap cl(\varphi))\}$
      if $(\mathbf{q} \cap cl(\varphi), X(\mathbf{q})) \notin unp \wedge \mathbf{q} \cap cl(\varphi) \notin S$
         then $unp := unp \cup \{(\mathbf{q} \cap cl(\varphi), X(\mathbf{q}))\}$

Note that states are restricted to be subsets of the closure of the initial formula. This is not essential, but guarantees that the automaton built by the procedure above is a subset of the one built by the abstract construction of Section 4. Thus, it only accepts words also accepted by this automaton. That it

[3] These sets are not strictly subsets of the closure since rules 1c and 1d can generate formulas that are elements of the closure preceded by the $\bigcirc$ operator.

accepts all words accepted by this automaton is a direct consequence of the fact that the *saturate* operation generates minimal sets (it only includes formulas that *must* be present) and of the argument used to justify the **omit transitions** simplification rule. The only somewhat delicate point concerns the special requirement on eventuality formulas imposed by the **omit transitions** rule. But, starting with a set containing an eventuality $e(\varphi')$, the *saturate* procedure always generates a set containing φ', the presence of suitable accepting states is thus guaranteed.

Example 7. Applying the *build-auto* algorithm to $\diamond p$, will produce the automaton in the following stages.

1. First, the initial states $\{\diamond p, p\}$ and $\{\diamond p\}$ are produced, with *unp* set to $\{(\{\diamond p, p\}, \emptyset), (\{\diamond p\}, \{\diamond p\})\}$.
2. $(\{\diamond p\}, \{\diamond p\})$ is removed from *unp* and transitions labeled by p and $\emptyset$ are created from the states$\{\diamond p\}$ to itself and to the state $\{\diamond p, p\}$.
3. $(\{\diamond p, p\}, \emptyset)$ is removed from *unp* and a transition labeled p from the $\{\diamond p, p\}$ state to the state $\emptyset$ is added; $(\emptyset, \emptyset)$ is added to *unp*.
4. $(\emptyset, \emptyset)$ is removed from *unp* and transitions labeled by p and $\emptyset$ are created from the state $\emptyset$ to itself.

5.3 Identifying equivalent states

One rather obvious limit of the "by need" procedure we have outlined, is that it only identifies states that are syntactically identical, i.e. consist of exactly the same set of formulas. A further reduction in the size of the automaton can thus be obtained by attempting to identify states that are semantically identical, i.e. that define identical sets of temporal sequences. Of course, deciding semantical equivalence in general is as hard as building an automaton from a temporal logic formula, and cannot be usefully used during the construction. However, one can identify some common semantical equivalences that can substantially reduce the size of the automaton. The following have, for instance, been used successfully [GPVW95,DGV99,SB00].

- $\{\varphi_1, \varphi_2, \varphi_1 \wedge \varphi_2\} \Leftrightarrow \{\varphi_1, \varphi_2\}$
- $\{\varphi_1, \varphi_1 \vee \varphi_2\} \Leftrightarrow \{\varphi_1\}$
- $\{\varphi_2, \varphi_1 \vee \varphi_2\} \Leftrightarrow \{\varphi_2\}$
- $\{\varphi_2, \varphi_1 \, U \, \varphi_2\} \Leftrightarrow \{\varphi_2\}$

The only caveat while using such semantical equivalences, is that they might change the definition of accepting states. One easy way around this is to disallow using such semantical equivalences involving formulas that are the argument of eventualities.

5.4 Further improvements

There are a number of further improvements that can be made to the construction of an automaton from a temporal logic formula. We briefly describe a few.

Simplifying the formula. Before applying the construction, it can be useful to rewrite the formula using some equivalence preserving transformation. For instance, one can use $\bigcirc \Box \Diamond \varphi \equiv \Box \Diamond \varphi$ to remove a "next" operator from a formula.

Early detection of inconsistencies. With the procedure we have outlined so far, inconsistencies are only detected after being propagated to the propositional level. Clearly, if a state contains both the formulas φ_1 and $\neg\varphi_1$ it is inconsistent and no transitions need leave that state.

Moving propositions from states to transitions. Propositional constraints are included in the states and uniformly applied to all transitions leaving a given state. In many cases this is wasteful and leads to excessively nondeterministic automata. An alternative is to let the choice of next state be determined by the propositions that are actually received as input. This allows propositional requirements to be removed from, and moved exclusively to, the transitions. However, special attention must be paid to the case in which propositions are the argument of eventualities. Indeed, removing them from states will then impact the definition of acceptance conditions.

Simplifying the acceptance condition. It is not uncommon for the structure of the strongly connected components of the automaton to allow a simplifying of the acceptance condition. This can lead to improved efficiency in the use of the automaton.

6 Conclusions

The intrinsically exponential complexity of building automata from temporal logic formulas, has long been seen as a limiting factor to the use of linear-time model checking. However, this conclusion ignores two important facts. First the formulas used in specifications are almost always very short. Second, the worst-case complexity bounds on building automata from temporal logic formulas are just that: *worst-case* bounds. The work surveyed in this paper shows that in the very large majority of cases, it is possible to build automata of very reasonable size for any temporal formula one would care to write in a specification. Of course, it is possible to produce pathological cases. For instance, using n propositions, one can polynomially encode an n bit counter in temporal logic, the corresponding automaton necessarily being exponential in n.

However, if one fixes the number of propositions, it is much less obvious to build a family of formulas for which the size of the corresponding automaton unavoidably exhibits exponential growth. In the opinion of this author, it is even highly unlikely that one would come upon such formulas when specifying program properties. The doubtful reader is invited to attempt to construct such a family of formulas.

So, in conclusion, it can be said that the work on improving the practical behavior of algorithms for generating automata from temporal logic formulas [GPVW95], [DGV99], [SB00] has been successful to the point of showing that the inherently exponential nature of the problem is of little practical significance.

This can be viewed as meaning that the lower bound proofs rely on using the expressive power of temporal logic in a way that is too unnatural to occur in many applications.

References

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.

[BVW94] Orna Bernholtz, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Stanford, California, June 1994. Springer-Verlag.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[DGV99] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *Computer-Aided Verification, Proc. 11th Int. Conference*, volume 1633, pages 249–260, July 1999.

[EC82] E.A. Emerson and E.M. Clarke. Using branching time logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[GPVW95] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. 15th Work. Protocol Specification, Testing, and Verification*, Warsaw, June 1995. North-Holland.

[Hol91] G. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International Editions, 1991.

[Hol97] Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997. Special Issue: Formal Methods in Software Practice.

[KV97] O. Kupferman and M. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.

[MW84] Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, January 1984.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th Int'l Symp. on Programming*, volume 137, pages 337–351. Springer-Verlag, Lecture Notes in Computer Science, 1981.

[Saf88] S. Safra. On the complexity of omega-automata. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, White Plains, October 1988.

[SB00] F. Somenzi and R. Bloem. Efficient büchi automata from ltl formulae. In *Computer-Aided Verification, Proc. 12th Int. Conference*, volume 1633, pages 247–263, 2000.

[SVW87] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science – Volume B: Formal Models and Semantics*, chapter 4, pages 133–191. Elsevier, Amsterdam, 1990.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *Proc. 24th IEEE Symposium on Foundations of Computer Science*, pages 185–194, Tucson, 1983.