Lecture Notes on Automated Verification

Deepak D'Souza
Dept. of Computer Science and Automation
Indian Institute of Science, Bangalore.
deepakd@csa.iisc.ernet.in

February 5, 2014

1 Transitions Systems and Automata

As a general theme in this course, labelled transition systems play the role of system models, while automata will represent specifications of required behaviour. In this section we introduce transition systems and classical automata and show how we do automated reasoning in this framework.

An *alphabet* is a finite set of symbols. For example $\Sigma = \{a, b, c\}$ is an alphabet. A *word* over an alphabet Σ is a finite sequence of symbols from Σ, denoted for example as *abbc*, *baa* etc. The empty word is denoted by ϵ. The set of all words over Σ is denoted by Σ^*.

A *language* over an alphabet Σ is a subset of Σ^*. For example $L_1 = \{abbc, baa\}$ is a finite language over $\Sigma = \{a, b, c\}$. $L_2 = \{\epsilon, ab, aabb, \ldots\}$, and $L_3 = \{a^i \mid i \text{ is prime}\}$ are infinite languages over Σ.

A *(labelled) transition system* over an alphabet Σ is a tuple $T = (Q, s, \rightarrow)$ where:

- Q is a finite set of states
- $s \in Q$ is the initial state
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation

We will view a transition system as a *generator* of words. Let $\mathcal{T} = (Q, s, \rightarrow)$ be a transition system over Σ. A *run* of $\mathcal{T}$ on a word $w = a_1 \cdots a_n \in \Sigma^*$ (with $n \geq 0$) is a finite sequence of states $q_0 q_1 q_2 \cdots q_n$, such that

- $q_0 = s$, and
- for each $i : \ 0 \leq i \leq n-1$, $q_i \stackrel{a_i+1}{\rightarrow} q_{i+1}$.

The language generated by $\mathcal{T}$ is defined to be

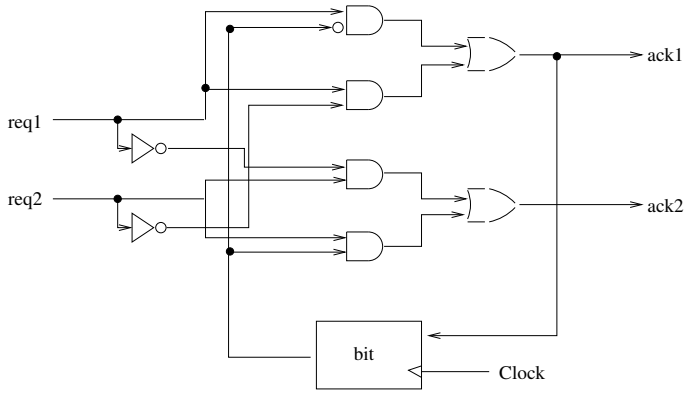
$$L(\mathcal{T}) = \{w \in \Sigma^* \mid \mathcal{S} \text{ has a run on } w\}.$$

It is convenient to sometimes use a *state-labelled* transition system. This is of the form $\mathcal{T} = (Q, S, \rightarrow l)$ where S is now a non-empty set of initial states, $\rightarrow Q \times Q$ is an (un-labelled) transition relation, and $l : Q \rightarrow \Sigma$ is a state-labelling function. A *run* of $\mathcal{T}$ on a word $w = a_1 \cdots a_n \in \Sigma^*$ $(n \geq 0)$ is a finite (possibly empty) sequence of states $q_1 q_2 \cdots q_n$, such that

- If $n \geq 1$ then $q_1 \in S$, and
- for each $i : \ 0 \leq i \leq n-1$, $q_i \rightarrow_{i+1}$ and for each $i : \ 1 \leq i \leq n$, $l(q_i) = a_i$.

Both forms of transition systems accept prefix-closed regular languages.

As an example we show how we can model a synchronous sequential circuit as a state-labelled transition system. This circuit is taken from [2]. The circuit is meant to behave as an arbiter for a resource, say a memory bus in a processor. It has two input lines *req1* and *req2*, and two output lines *ack1* and *ack2*. A process asserts its request line when it needs the resource, and its acknowledgment line is asserted (by the arbiter) in case it is granted the resource. Below is a diagram of the circuit:



We assume the circuit is driven by a clock and the latch is edge-triggered. The circuit works in a latch–change-inputs–read-output cycle. The initial value of the latch and request values is assumed to be 0.

The circuit is modelled as a state-labelled transition system in Figure 1. Each state label is of the form $(req1, req2, ack1, ack2, bit)$.

Turning now to specifications of behaviours, we will use automata to describe specifications. In this section, we will look at "safety" specifications.

Recall that a (non-deterministic) finite state automaton (NFA) is a structure of the form $\mathcal{A} = (Q, S, \rightarrow F)$ similar to a transition-labelled transition system, except that S is a set of initial states, and F is a set of final states. A run of $\mathcal{A}$ on a word w is now accepting only if it ends in a state in F. The language generated by $\mathcal{A}$ is defined to be

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{S} \text{ has an accepting run on } w\}.$$

A specification of *safe* behaviours (or a *safety* specification) is simply an automaton which accepts a prefix-closed language (or equivalently a transition

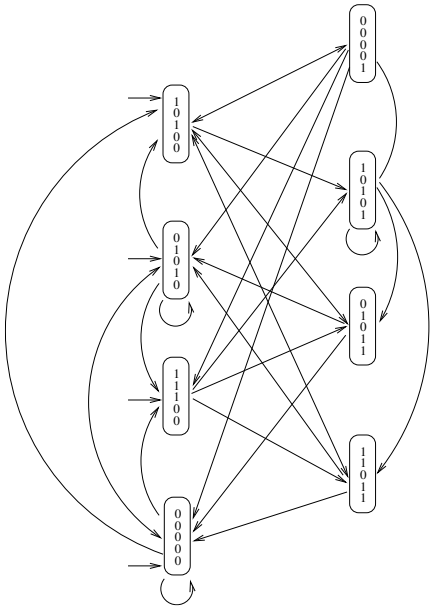


Figure 1: Transition system for the arbiter circuit.

system). If we view the set of all words over an alphabet Σ as a downward growing tree (as shown in Fig 2) then a safety specification can be viewed as a "sub-cone" of this tree. A system modelled as a transition system $\mathcal{T}$ over Σ is said to satisfy a safety spec $L(\mathcal{A})$ iff $L(\mathcal{T}) \subseteq L(\mathcal{A})$.

- Give example of specs for example above.
- Other examples of specs
- Solving the verification problem. Complexity (refer to exercise).

2 Verification with Buchi Automata

Automata on infinite words were introduced by Büchi and Muller in the 1960's with a view to solving problems in Logic and switching theory respectively. Our interest however is in using these automata as specifications of infinite behaviours.

Why do we consider *infinite* (non-terminating) behaviours at all? Simply because this is a natural thing to do when we want to reason about certain properties of "reactive" systems. In particular, "liveness" properties like "a request is eventually granted", or in general, "something good *eventually* happens". When the eventuality is bounded, as in "a request is granted within 5 steps", then we can use classical automata on finite words to describe the good behaviours we expect to see. However, when we have no bound in our specification, it is not clear how one can specify such a property as a property of the

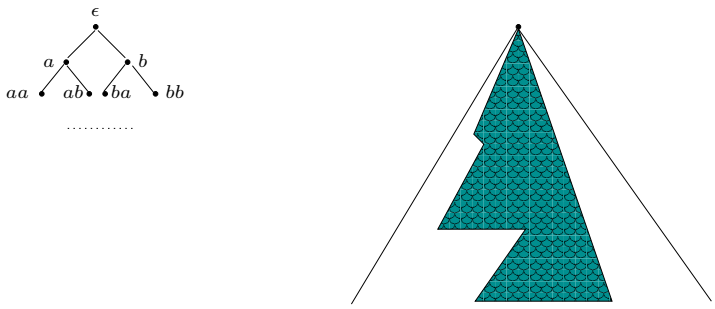


Figure 2: Tree of words over $\Sigma = \{a, b\}$ and a safety spec.

finite behaviours that the system can generate. On the other hand, it is much more natural to specify that all infinite runs of the system satisfy the property that every request is eventually granted.

Though Linear-time Temporal Logic (LTL) is a more standard way of specifying infinite behaviours, we use Büchi automata to model specifications for the following reasons:

- Büchi automata are technically more expressive than LTL. Thus we could potentially specify properties like "even number of a's and b's between c's" which cannot be specified in LTL (though in practice these properties may not be very useful).
- We will solve the LTL model-checking problem later using the automata-theoretic approach, where we translate an LTL formula into a corresponding Büchi automaton. Thus the verification framework for Büchi automata can be used there too.
- Some verification tools like Spin (and Mona?) allow specifications of arbitrary Büchi automata.

[Material from [3]] - Introduction to Büchi automata, definitions, examples.

To follow the model-checking framework for classical automata, we need to be able to complement Büchi automata, intersect it with the system, and check the combined automaton for emptiness.

- Answering questions about automata: emptiness, closure under union, intersection, and complementation.

Here is a characterisation of ω-regular languages along the lines of Kleene's regular expressions, via ω-regular expressions:

Theorem 2.1 *ω-regular languages (i.e. those accepted by Büchi automata) can be characterised as finite unions of ω-languages of the form $U \cdot V^\omega$, where U and V are regular languages of finite words.*

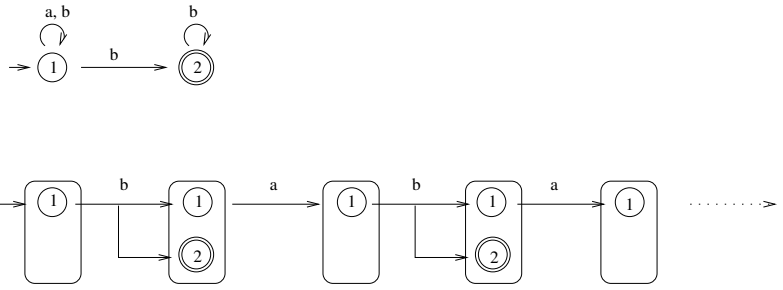


Figure 3: an "accepting" run of the "subset automaton".

Proof Let $\mathcal{A} = (Q, s, \rightarrow, F)$ be a Büchi automaton. Let $W_{pq} = \{w \mid p \xrightarrow{w} q\}$. Then

$$L(\mathcal{A}) = \bigcup_{f \in F} (W_{sf} \cdot (W_{ff})^{\omega})$$

Conversely, we note that if U is regular, and L is ω-regular, then U^{ω} and $U \cdot L$ are both ω-regular. □

We give a fleshed-out version of the proof of complemenation from [3]. Before that, we try to see why the classical "subset construction" fails for determinizing non-deterministic Büchi automata. (One may ask "what if we could? how do we complement a deterministic Büchi automaton?" This can be done by guessing a point after which we see no more F states and going over to a copy in which all F states are deleted, and all non-F states are now final.)

Consider the non-deterministic Büchi automaton accepting the language of finitely many a's given in Fig 2. Suppose we construct the subset automaton and take the set of subset states that contain a final state of the original automaton as the Büchi accepting states. Consider the run of the subset automaton on $(ab)^{\omega}$ as shown. The word $(ab)^{\omega}$ is accepted as it sees a "final" subset state infinitely often. However there is no infinite run of the original automaton along these subset states that visits state 2 infinitely often.

As it turns out, deterministic Büchi automata are *strictly* less expressive than their non-deterministic counterparts. (The above exercise is not entirely futile as Safra shows how one can use essentially the same subset construction – though with a *Rabin* condition – to get a deterministic version of the given Büchi automaton).

Here is a characterisation of languages accepted by deterministic Büchi automata: they are all *limits* of a regular language.

$$lim(W) = \{\alpha \mid \alpha \text{ has infinitely many prefixes in } W\}$$

Proof : Given $\mathcal{A}$ show that $L_{\omega}(\mathcal{A}) = lim(L_f(\mathcal{A}))$. □

We can now show that L over $\{a, b\}$ of finitely many a's is not definable by a deterministic Büchi automaton. If so then $L = lim(W)$ for some regular W.

Now since $ab^\omega \in L$, we have $ab^{i_0} \in W$ for some i_0. Similarly, $ab^{i_0}ab^\omega \in L$, hence $ab^{i_0}ab^{i_1} \in W$ for some i_1. Continuing in this way, we can conclude that L must contain the ω-word $\alpha = ab^{i_0}ab^{i_1}\cdots$, since it has infinitely many prefixes in W and hence belongs to $lim(W)$, which is L. This is a contradiction, since α has infinitely many a's.

We can have other acceptance conditions which preserve the class of languages defined by non-deterministic Büchi automata, but which admit determinization.

Muller: We have an underlying transition system $(Q, s, \longrightarrow)$ as before, along with a set of subsets of Q: $\{F_1, \ldots, F_k\}$, with each $F_i \subseteq Q$. A run ρ is accepting according to this condition iff $inf(\rho) = F_i$ for some i.

Rabin: A set of pairs of subsets of Q: $\{(R_1, G_1), (R_2, G_2), \ldots (R_k, G_k)\}$. A run ρ is accepting according to such a Rabin condition iff for some i, $inf(\rho) \cap R_i = \emptyset$ and $inf(\rho) \cap G_i \neq \emptyset$.

Street (this is the negation of the Rabin condition): A set of pairs of subsets of Q: $\{(R_1, G_1), (R_2, G_2), \ldots (R_k, G_k)\}$, with a run ρ being accepting iff for all i, $inf(\rho) \cap R_i \neq \emptyset$ or $inf(\rho) \cap G_i = \emptyset$.

It is easy to see that BA = MA = RA = SA and that the translation from BA to MA, and RA to MA, both preserve the structure of the automata (in particular if the automaton is deterministic to begin with, its translation to MA is also deterministic). In general, however, going from MA to BA or RA introduces non-determinsm.

Routes for complementation:

- Safra shows that it is possible to extend the idea of the subset construction to determinize an Büchi automaton into a deterministic Rabin automaton. The same automaton viewed as a Street automaton recognizes the complement of the original Büchi automaton. We can now convert the deterministic Street automaton to a non-deterministic Büchi automaton. This can be done by taking the intersection of the Büchi automata corresponding to each (R_i, G_i) pair. The Büchi automaton simply checks if either an R_i state is seen infinitely often, or – after a point – the G_i states are never seen. Note that intersection works here because the automaton was deterministic to begin with. Alternately, one can use a more efficient construction due to Vardi (see [1]).

- McNaughton showed that Buchi automata are equivalent in expressive power to deterministic Muller automata. Since deterministic Muller automata can be easily complemented, we can now translate the complemented Muller automaton to a non-deterministic Büchi automaton.

  Note that Safra's construction gives an alternate proof of McNaughton's result: Given a MA we can go to a (non-det) BA, from there to a deterministic RA (using Safra's construction), and (trivially) back to a deterministic MA.

- Büchi: using congruences. We describe this below.

Büchi's procedure for complementation.

Given $\mathcal{A} = (Q, s, \longrightarrow, F)$ a Büchi automaton over Σ. We proceed as follows: Show that we can define a congruence $\sim_{\mathcal{A}}$ of finite index on Σ^* with the following properties. Let S be the set of equivalence classes of $\sim_{\mathcal{A}}$. Then

1. For any $U, V \in S$, if $L(\mathcal{A}) \cap U \cdot V^\omega \neq \emptyset$ then $U \cdot V^\omega \subseteq L(\mathcal{A})$.
2. $\Sigma^\omega = \bigcup_{U,V \in S} U \cdot V^\omega$.

It follows that both $L(\mathcal{A})$ and $\Sigma^\omega - L(\mathcal{A})$ are finite unions of sets of the form $U \cdot V^\omega$, for $U, V \in S$. This proves that ω-regular languages are closed under complement. To see that this gives us an effective procedure for complementing $\mathcal{A}$, we observe that the languages U in S are regular languages (being equivalence classes of a congruence of finite index) and in fact computable (i.e. we can construct automata for each of the languages in S (Exercise!)). Hence we can find all the $U.V^\omega$ which have an empty intersection with $L(\mathcal{A})$ and take their union to get an automaton accepting $\Sigma^\omega - L(\mathcal{A})$.

- Define $\sim_{\mathcal{A}}$.
- Show that $|S| \leq 2^{2.n^2}$. Every equivalence class has a unique pair (X, Y) associated with it, where X and Y are subsets of $2^{Q \times Q}$, representing respectively the exact set of pairs (q, q') on which there is a path on all u in the equivalence class from q to q', and the set of pairs on which there is a path that visits F. The number of such pairs (X, Y) is at most $2^{2.n^2}$.

To prove the first part of the claim, let $\alpha \in U.V^\omega \cap L(\mathcal{A})$, and let $\beta \in U.V^\omega$. Then $\alpha = u \cdot v_0 \cdot v_1 \cdots$ and $\beta = u' \cdot v_0' \cdot v_1' \cdots$ for some $u, u' \in U$ and $v_i, v_i' \in V$. An accepting run ρ of $\mathcal{A}$ on α now easily gives rise to an accepting run ρ' of $\mathcal{A}$ on β, using the properties of $\sim_{\mathcal{A}}$.

To prove the second part of the claim: Let $\alpha \in \Sigma^\omega$. We want to break up α into $u \cdot v_1 \cdot v_2 \cdots$ with the v_i's $\sim_{\mathcal{A}}$-equivalent. Refer to figure 4.

- Define an equivalence on positions of α as follows: $k \sim k'$ iff there exists some $m \geq k, k'$ with $\alpha[k, m] \sim_{\mathcal{A}} \alpha[k', m]$ (in this case we say k and k' *merge* at m).
- $\sim$ is of finite index over $\mathbb{N}$.

  If not, $\sim$ has infinitely many positions k_i, no two of which merge at any point. Let $\sim_{\mathcal{A}}$ have n equivalence classes. Consider the positions $k_0, \ldots, k_n$. At the postion $k_n + 1$, the words $\alpha[k_i, k_n + 1]$ for each $i \in \{0, \ldots, n\}$ belong to distinct equivalence classes of $\sim_{\mathcal{A}}$. But this is a contradiction.
- So there is an equivalence class of $\sim$ that repeats infinitely often along α. Let these positions be $k_0, k_1, \ldots$.

  Now there is a subsequence of $< k_i >$, say $< l_i >$ which has the property that $\alpha[l_0, l_i] \sim_{\mathcal{A}} \alpha[l_0, l_j]$ for all i, j. This is true since, if we consider the

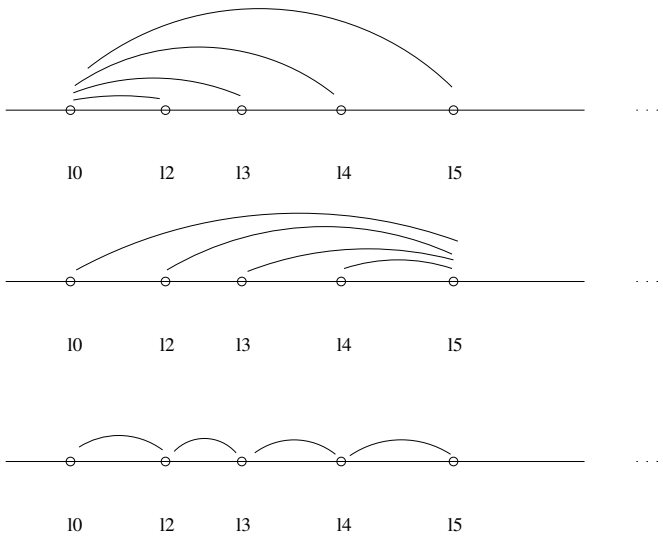


Figure 4: Arguing "completeness"

position k_0, the $\sim_{\mathcal{A}}$ equivalence classes it sees at the subsequent k_i's are finite, and hence there must be one class which repeats infinitely often. Take these positions as the l_i's.

- We can further assume that the l_i's are such that each of $l_0, \ldots, l_i$ merge at l_{i+1}, for each i.
- This now lets us argue that the $\alpha[l_i, l_{i+1}]$'s are all $\sim_{\mathcal{A}}$-equivalent.
- Mention that size of complemented automaton is $2^{4.n^2}$. It seems easy to see it is $2^{O(n^2)}$.

3 Linear-Time Temporal Logic

Motivation for using LTL: simpler to use than automata, fairly natural and concise for most specifications.

We consider the following syntax of LTL(P) parameterised by a countable set of propositions P.

$$\varphi ::= \top \mid \ p \mid O\varphi \mid (\varphi U \varphi) \mid \Diamond\varphi \mid \Box\varphi \mid \neg\varphi \mid (\varphi \vee \varphi)$$

Here $p \in P$.

The logic is interpreted over infinite sequences of "worlds" or propositional valuations for P. We represent a propositional valuation as a subset v of P, in which the variables in v are set to true, and the rest to false. A model for a

formula φ in the logic is thus of the form $\alpha \in (2^P)^\omega$. The satisfaction relation, $\alpha \models \varphi$ ("$\alpha models \varphi$") is given by:

$$
\begin{array}{lll}
\alpha, i \models \top & & \\
\alpha, i \models p & \text{iff} & p \in \alpha(i) \\
\alpha, i \models O\varphi & \text{iff} & \alpha, i+1 \models \varphi \\
\alpha, i \models \varphi U \eta & \text{iff} & \exists k \geq i : \alpha, k \models \eta \text{ and } \forall j : i \leq j < k,\ \alpha, j \models \varphi \\
\alpha, i \models \Diamond\varphi & \text{iff} & \exists k \geq i : \alpha, k \models \varphi \\
\alpha, i \models \Box\varphi & \text{iff} & \forall k \geq i : \alpha, k \models \varphi \\
\alpha, i \models \neg\varphi & \text{iff} & \alpha, i \not\models \varphi \\
\alpha, i \models \varphi \vee \psi & \text{iff} & \alpha, i \models \varphi \text{ or } \alpha, i \models \psi
\end{array}
$$

The modalities $\Diamond$ and $\Box$ are expressible using U: $\Diamond\varphi = \top U \varphi$, and $\Box\varphi = \neg(\top U \neg\varphi)$. We say $\alpha \models \varphi$ if $\alpha, 0 \models \varphi$. We denote by $L(\varphi)$ the set $\{\alpha \in (2^P)^\omega \mid \alpha \models \varphi\}$.

Examples: for mutual exclusion: safety: $\Box(\neg(1atl_3 \wedge 2atl_3))$ and liveness: $\Box((1atl_1 \Rightarrow \Diamond 1atl_3) \wedge (2atl_1 \Rightarrow \Diamond 2atl_3))$. For alternating bit protocol: fairness (messages are delivered infinitely often implies sent message eventually reaches).

Satisfiability and Model-checking problems: The *satisfiability* problem for LTL is given an LTL formula φ is there a model α such that $\alpha \models \varphi$?

In the *model-checking* problem we are given a finite state program modelled as a state-labelled transition system $\mathcal{T}$, where the labels come from 2^P, and an LTL(P) formula φ, and we have answer whether $\mathcal{T} \models \varphi$ in the sense that every run of $\mathcal{T}$ satisfies φ: more precisely is $L(\mathcal{T}) \subseteq L(\varphi)$?

Both problems are equivalent in a sense (Exercise!).

Both problems can be solved if we can show that we can construct for any given LTL(P) formula, a Büchi automaton $\mathcal{A}_\varphi$ over the alphabet 2^P, accepting precisely the models of φ: i.e. $L(\mathcal{A}_\varphi) = L(\varphi)$.

Some examples of manually constructing BA for given LTL formulas: $\Diamond(pUq)$, $\Box(pUq)$.

A natural approach to try: inductively associate a BA with each subformula. This works fine for p, Op, and boolean combinations. But for $\varphi U \psi$ it is not easy. We can guess a point where ψ will be true, and spawn off a copy of $\mathcal{A}_\psi$ to verify that. However, we need to check that φ is true at *each* point till then, and this would require spawning off copies of $\mathcal{A}_\varphi$ at all these points. Thus, maybe with a theory of *alternating* Büchi automata one could pull this off.

We now sketch the technique due to Vardi and Wolper (construction of the Vardi-Wolper formula automaton). Idea: use state to specify exacly the sub-formulas we are going to satisfy, use a "two-state" (current-state and next-state) semantics for the temporal operators, and use the transition relation of the automaton to locally ensure this, and use the Büchi acceptance condition to ensure that eventualities like μ in $\psi U \mu$ are met.

Let φ be an LTL formula over the set of propositions P. We define a Büchi automaton $\mathcal{A}_\varphi$ over 2^P which accepts precisely the set of models of φ.

Let $sfc(\varphi)$ be the set of subformulas of φ. We define $cl(\varphi)$, the Fisher-Ladner

closure of a formula φ, to be

$$cl(\varphi) = X \cup \{\neg\beta \mid \beta \in X\},$$

where $X = sfc(\varphi) \cup \{O(\psi U \mu) \mid \psi U \mu \in sfc(\varphi)\}$

Define an *atom* of φ to be a maximally consistent subset of $cl(\varphi)$. Formally, a subset A of $cl(\varphi)$ is an *atom* of φ iff

1. $\forall \psi \in cl(\varphi)$, $\neg\psi \in A$ iff $\psi \notin A$. (Here we identify $\neg\neg\psi$ with ψ.)
2. $\forall (\psi \vee \mu) \in cl(\varphi)$, $(\psi \vee \mu) \in A$ iff $\psi \in A$ or $\mu \in A$.
3. $\forall (\psi U \mu) \in cl(\varphi)$, $(\psi U \mu) \in A$ iff $\mu \in A$, or, both ψ, $O(\psi U \mu) \in A$.

We can now define the automaton $\mathcal{A}_\varphi^{\text{LTL}} = (Q, Q_0, \longrightarrow, \mathcal{F})$. We use a *generalized* Büchi condition which is of the form $\mathcal{F} = \{F_1, \ldots, F_m\}$. A run $\rho : \mathbb{N} \to Q$ is accepting according to $\mathcal{F}$ iff for each $i \in \{1, \ldots, m\}$, $\rho(j) \in F_i$ for infinitely many $j \in \mathbb{N}$. A generalized Büchi condition can be easily converted to a Büchi condition by augmenting the states with a 0–k counter.

- Take Q to be the set of atoms of φ.
- Q_0 is the set of atoms in Q which contain φ.
- The transition relation $\to e$ is given by the following rule. We have $A \xrightarrow{v} B$ iff each of the following is satisfied:
  1. $v = A \cap P$.
  2. For each $O\psi \in cl(\varphi)$, $O\psi \in A$ iff $\psi \in B$.
- For the final states we have a generalized Büchi condition $\mathcal{F} = \{F_1, \ldots, F_m\}$ where $m \geq 0$ is the number of until formulas in $cl(\varphi)$. Let $\{\psi_1 U \mu_1, \ldots, \psi_m U \mu_m\}$ be the set of until formulas in $cl(\varphi)$. Then for each $i \in \{1, \ldots, m\}$ we define $F_i = \{A \mid \psi_i U \mu_i \notin A \text{ or } \mu_i \in A\}$.

Example of $p\ U q$.
Claim: $L(\mathcal{A}) = L(\varphi)$.
Proof: soundess and completeness.

References

[1] [M96]Madhavan Mukund: Automata on infinite inputs, Technical Report TCS-96-2, Chennai Mathematical Institute (1996).

[2] [M98]K. L. McMillan, Getting started with SMV, 1998.

[3] [T90]W. Thomas: Automata on Infinite Objects, in J. V. Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, Vol. B, 133–191, Elsevier Science Publ., Amsterdam (1990).