Part 4: Büchi automata

Preview

Model-checking problem: $[\![\mathcal{K}]\!] \subseteq [\![\phi]\!]$ – how can we check this algorithmically?

(Historically) first approach: Translate $\mathcal{K}$ into an LTL formula $\psi_{\mathcal{K}}$, check whether $\psi_{\mathcal{K}} \rightarrow \phi$ is a tautology. Problem: very inefficient.

Language-/automata-theoretic approach: $[\![\mathcal{K}]\!]$ and $[\![\phi]\!]$ are languages (of infinite words).

Find a suitable class of automata for representing these languages.

Define suitable operations on these automata for solving the problem.

This is the approach we shall follow.

Büchi automata

A Büchi automaton is a tuple

$$\mathcal{B} = (\Sigma, S, s_0, \Delta, F),$$

where:

Σ	is a finite alphabet;
S	is a finite set of states;
$s_0 \in S$	is an initial state;
$\Delta \subseteq S \times \Sigma \times S$	are transitions;
$F \subseteq S$	are accepting states.

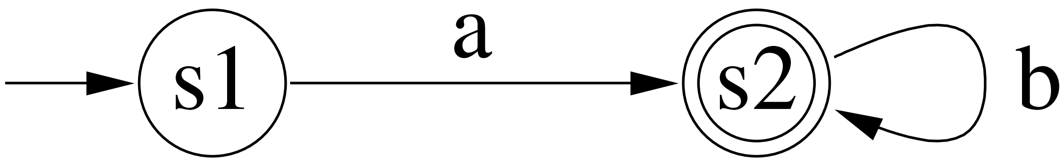
Remarks:

Definition and graphical representation like for finite automata.

However, Büchi automata are supposed to work on *infinite* words, requiring a different acceptance condition.

Example

Graphical representation of a Büchi automaton:



The components of this automaton are $(\Sigma, S, s_1, \Delta, F)$, where:

- $\Sigma = \{a, b\}$ (symbols on the edges)
- $S = \{s_1, s_2\}$ (circles)
- s_1 (indicated by arrow)
- $\Delta = \{(s_1, a, s_2), (s_2, b, s_2)\}$ (edges)
- $F = \{s_2\}$ (with double circle)

Language of a Büchi automaton

Let $\mathcal{B} = (\Sigma, S, s_0, \Delta, F)$ be a Büchi automaton.

A run of $\mathcal{B}$ over an infinite word $\sigma \in \Sigma^\omega$ is an infinite sequences of states $\rho \in S^\omega$ where $\rho(0) = s_0$ and $(\rho(i), \sigma(i), \rho(i+1)) \in \Delta$ for $i \geq 0$.

We call ρ accepting iff $\rho(i) \in F$ for infinitely many values of i.

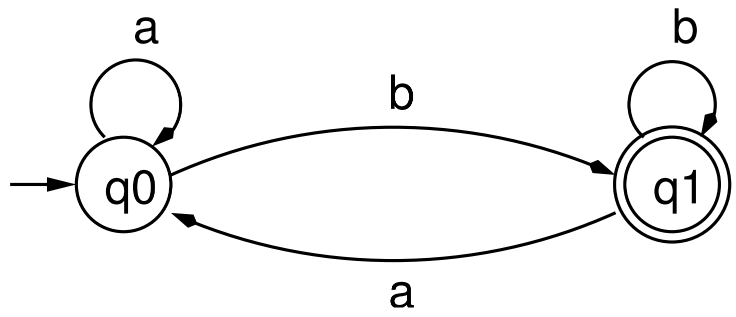
> I.e., ρ infinitely often visits accepting states.
> (By the pigeon-hole principle: at least one accepting state is visited infinitely often.)

$\sigma \in \Sigma^\omega$ is accepted by $\mathcal{B}$ iff there exists an accepting run over σ in $\mathcal{B}$.

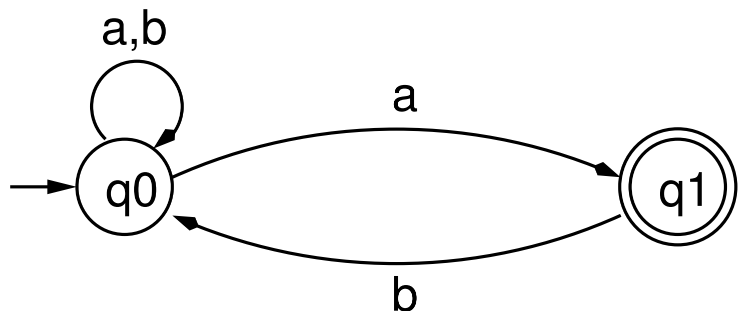
The language of $\mathcal{B}$, denoted $\mathcal{L}(\mathcal{B})$, is the set of all words accepted by $\mathcal{B}$.

Büchi automata: examples

“infinitely often b”



“infinitely often ab”



Büchi automata and LTL

Let AP be a set of atomic propositions.

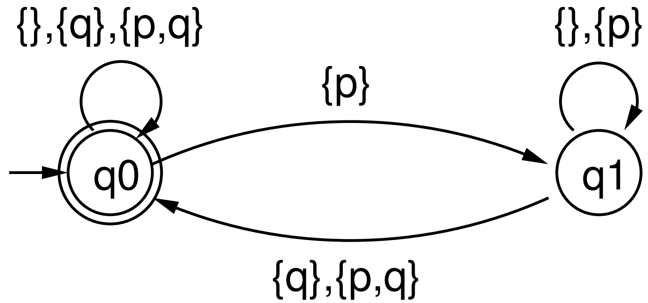
A Büchi automaton with alphabet 2^{AP} accepts a sequence of valuations.

Claim: For every LTL formula ϕ there exists a Büchi automaton $\mathcal{B}$ such that $\mathcal{L}(\mathcal{B}) = [\![\phi]\!]$.

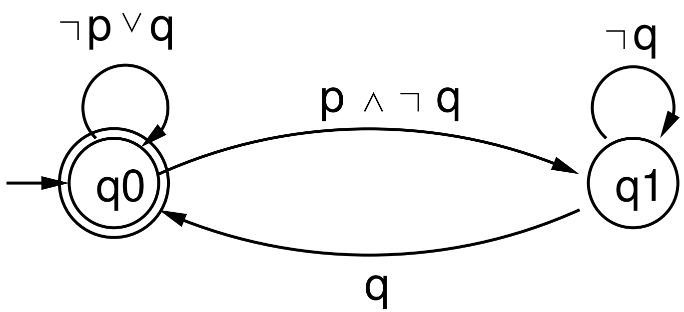
(We shall prove this claim later.)

Examples: $\mathbf{F}\, p$, $\mathbf{G}\, p$, $\mathbf{G}\,\mathbf{F}\, p$, $\mathbf{G}(p \to \mathbf{F}\, q)$, $\mathbf{F}\,\mathbf{G}\, p$

Example automaton for $\mathbf{G}(p \rightarrow \mathbf{F}\, q)$, with $AP = \{p, q\}$.



Alternatively we can label edges with formulae of propositional logic; in this case, a formula F stands for all elements of $[\![F]\!]$. In this case:



Operations on Büchi automata

The languages accepted by Büchi automata are also callled ω-regular languages.

Like the usual regular languages, ω-regular languages are also closed under Boolean operations.

I.e., if $\mathcal{L}_1$ and $\mathcal{L}_2$ are ω-regular, then so are

$$\mathcal{L}_1 \cup \mathcal{L}_2, \qquad \mathcal{L}_1 \cap \mathcal{L}_2, \qquad \mathcal{L}_1^c.$$

We shall now define operations that take Büchi automata accepting some languages $\mathcal{L}_1$ and $\mathcal{L}_2$ and produce automata for their union or intersection.

In the following slides we assume $\mathcal{B}_1 = (\Sigma, S, s_0, \Delta_1, F)$ and $\mathcal{B}_2 = (\Sigma, T, t_0, \Delta_2, G)$ (with $S \cap T = \emptyset$).

Union

"Juxtapose" $\mathcal{B}_1$ and $\mathcal{B}_2$ and add a new initial state.

In other words, the automaton $\mathcal{B} = (\Sigma,\ S \cup T \cup \{u\},\ u,\ \Delta_1 \cup \Delta_2 \cup \Delta_u,\ F \cup G)$ accepts $\mathcal{L}(\mathcal{B}_1) \cup \mathcal{L}(\mathcal{B}_2)$, where

- u is a "fresh" state ($u \notin S \cup T$);
- $\Delta_u = \{\, (u, \sigma, s) \mid (s_0, \sigma, s) \in \Delta_1 \,\} \cup \{\, (u, \sigma, t) \mid (t_0, \sigma, t) \in \Delta_2 \,\}$.

Intersection I (a special case)

We first consider the case where all states in $\mathcal{B}_2$ are accepting, i.e. $G = T$.

Idea: Construct a cross-product automaton (like for FA), check whether F is visited infinitely often.

Let $\mathcal{B} = (\Sigma,\ S \times T,\ \langle s_0, t_0\rangle,\ \Delta,\ F \times T)$, where

$$\Delta = \{\, (\langle s, t\rangle, a, \langle s', t'\rangle) \mid a \in \Sigma,\ (s, a, s') \in \Delta_1,\ (t, a, t') \in \Delta_2 \,\}.$$

Then: $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$.

Intersection II (the general case)

Principle: We again construct a cross-product automaton.

Problem: The acceptance condition needs to check whether *both* accepting sets are visited infinitely often.

Idea: create two copies of the cross product.

- In the first copy we wait for a state from F.
- In the second copy we wait for a state from G.
- In both copies, once we've found one of the states we're looking for, we switch to the other copy.

We will choose the acceptance condition in such a sway that an accepting run switches back and forth between the copies infinitely often.

Let $\mathcal{B} = (\Sigma, U, u, \Delta, H)$, where

$$U = S \times T \times \{1, 2\}, \quad u = \langle s_0, t_0, 1 \rangle, \quad H = F \times T \times \{1\}$$

$$(\langle s, t, 1 \rangle, a, \langle s', t', 1 \rangle) \in \Delta \text{ iff } (s, a, s') \in \Delta_1, \ (t, a, t') \in \Delta_2, \ s \notin F$$
$$(\langle s, t, 1 \rangle, a, \langle s', t', 2 \rangle) \in \Delta \text{ iff } (s, a, s') \in \Delta_1, \ (t, a, t') \in \Delta_2, \ s \in F$$
$$(\langle s, t, 2 \rangle, a, \langle s', t', 2 \rangle) \in \Delta \text{ iff } (s, a, s') \in \Delta_1, \ (t, a, t') \in \Delta_2, \ t \notin G$$
$$(\langle s, t, 2 \rangle, a, \langle s', t', 1 \rangle) \in \Delta \text{ iff } (s, a, s') \in \Delta_1, \ (t, a, t') \in \Delta_2, \ t \in G$$

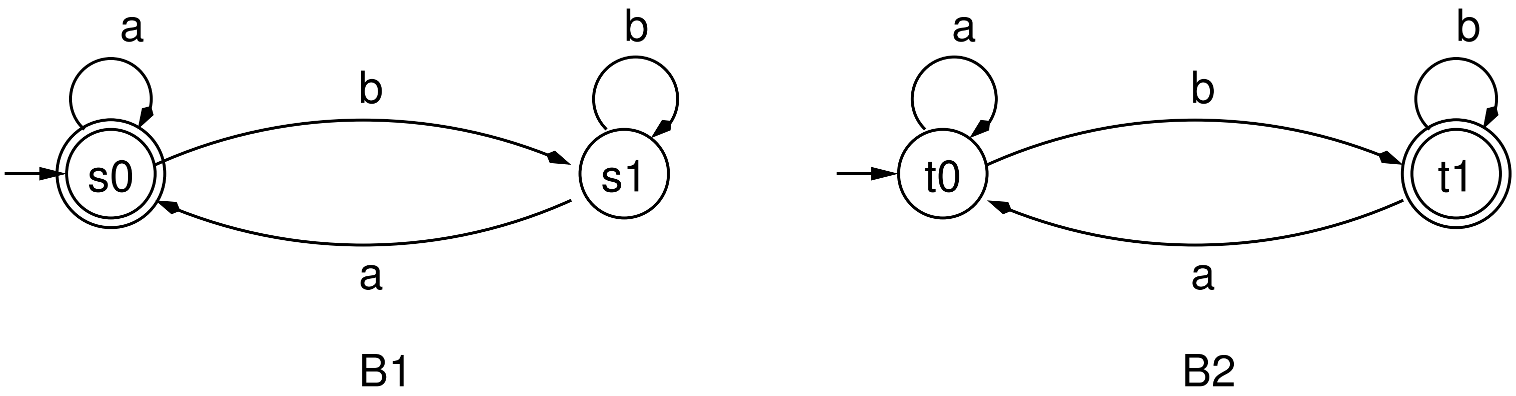
Remarks:

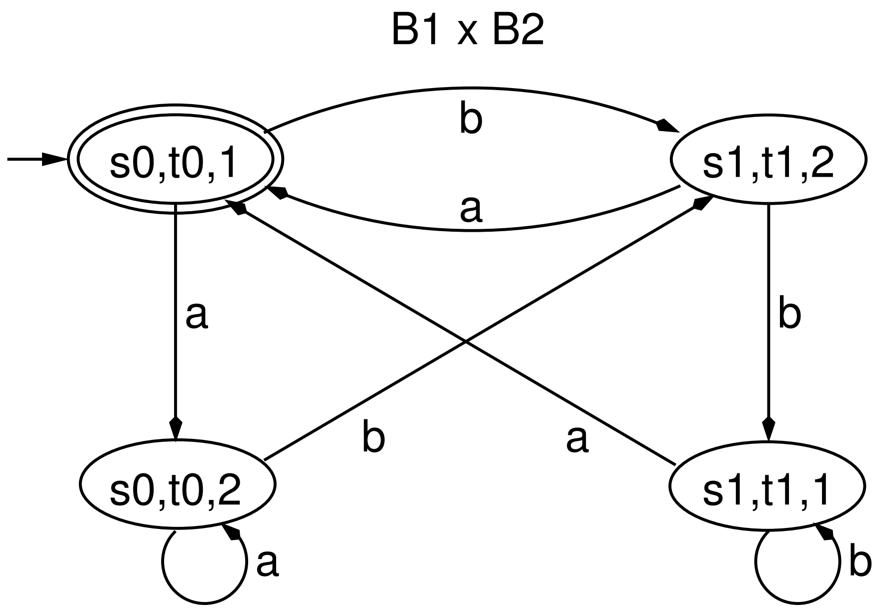
The automaton starts in the first copy.

We could have chosen other acceptance conditions such as $S \times G \times \{2\}$.

The construction can be generalized to intersecting n automata.

Intersection: example





Complement

Problem: Given $\mathcal{B}_1$, construct $\mathcal{B}$ with $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}_1)^c$.

Such a construction is possible (but rather complicated). We will not require it for the purpose of this course.

Additional literature:

Wolfgang Thomas, *Automata on Infinite Objects*,
Chapter 4 in *Handbook of Theoretical Computer Science*,

Igor Walukiewicz, lecture notes on *Automata and Logic*, chapter 3,
`www.labri.fr/Perso/~igw/Papers/igw-eefss01.ps`

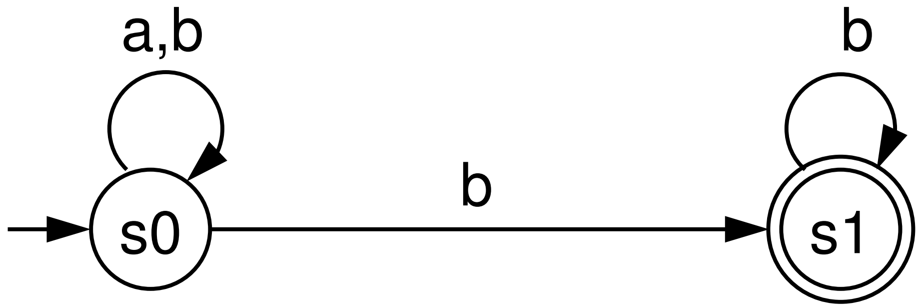
Deterministic Büchi automata

For finite automata (known from *regular language theory*), it is known that every language expressible by a finite automaton can also be expressed by a deterministic automaton, i.e. one where the transition relation Δ is a function $S \times \Sigma \to S$.

Such a procedure does not exist for Büchi automata.

In fact, there is no *deterministic* Büchi automaton accepting the same language as the automaton below:

“Only finitely many as.”



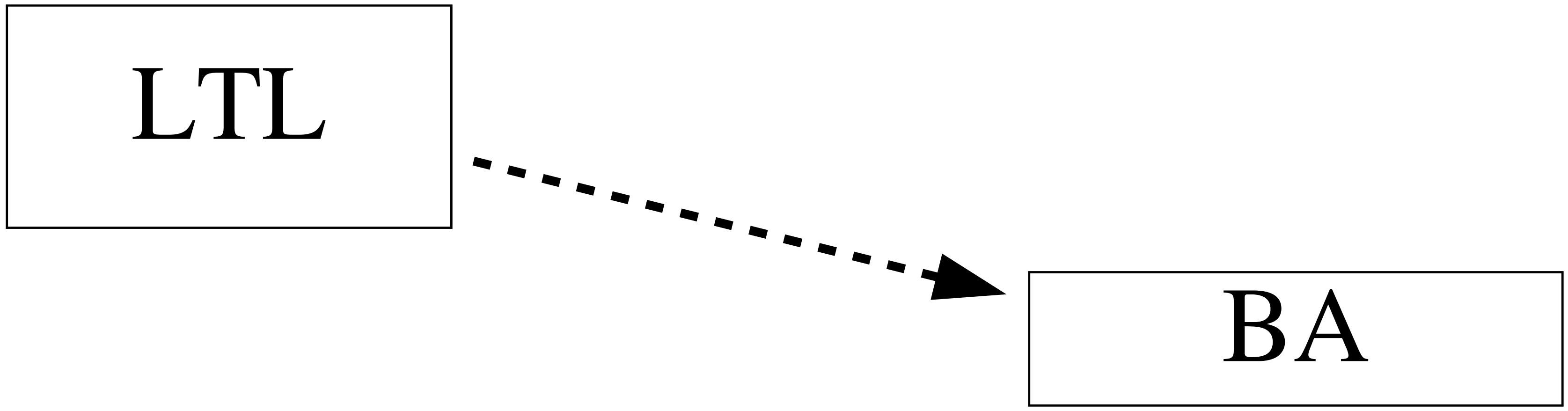
Proof: Let $\mathcal{L}$ be the language of infinite words over $\{a, b\}$ containing only finitely many as. Assume that a deterministic Büchi automaton $\mathcal{B}$ with $\mathcal{L}(\mathcal{B}) = \mathcal{L}$ exists, and let n be the number of states in $\mathcal{B}$.

We have $b^\omega \in \mathcal{L}$, so let α_1 be the (unique) accepting run for b^ω. Suppose that an accepting state is first reached after n_1 letters, i.e. $s_1 := \alpha_1(n_1)$ is the first accepting state in α_1.

We now regard the word $b^{n_1}ab^\omega$, which is still in $\mathcal{L}$, therefore accepted by some run α_2. Since $\mathcal{B}$ is deterministic, α_1 and α_2 must agree on the first n_1 states. Now, watch for the second occurrence of an accepting state in α_2, i.e. let $s_2 := \alpha_2(n_1 + 1 + n_2)$ be an accepting state for n_2 minimal. Then, $s_1 \neq s_2$ because otherwise there would be a loop around an accepting state containing a transition with an a.

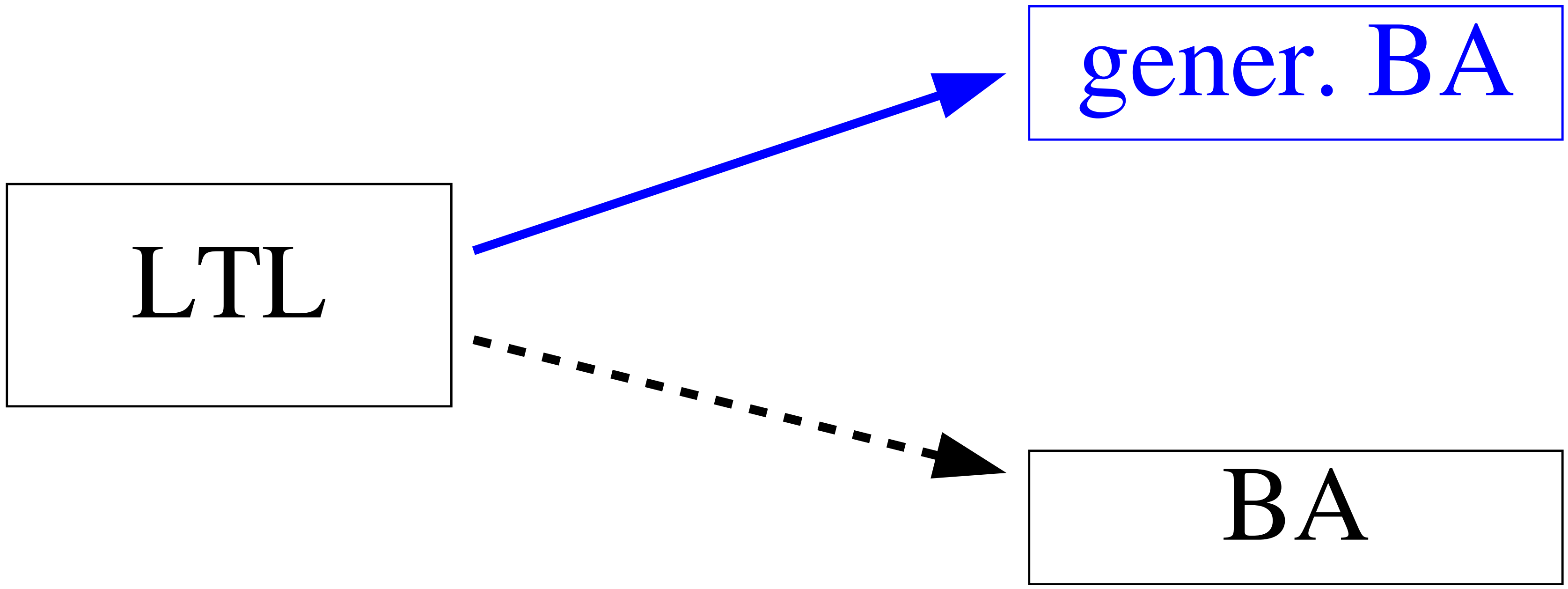
We now repeat the argument for $b^{n_1}ab^{n_2}ab^\omega$, derive the existence of a third distinct state, etc. After doing this $n + 1$ times, we conclude that $\mathcal{B}$ must have more than n distinct states, a contradiction.

Preview



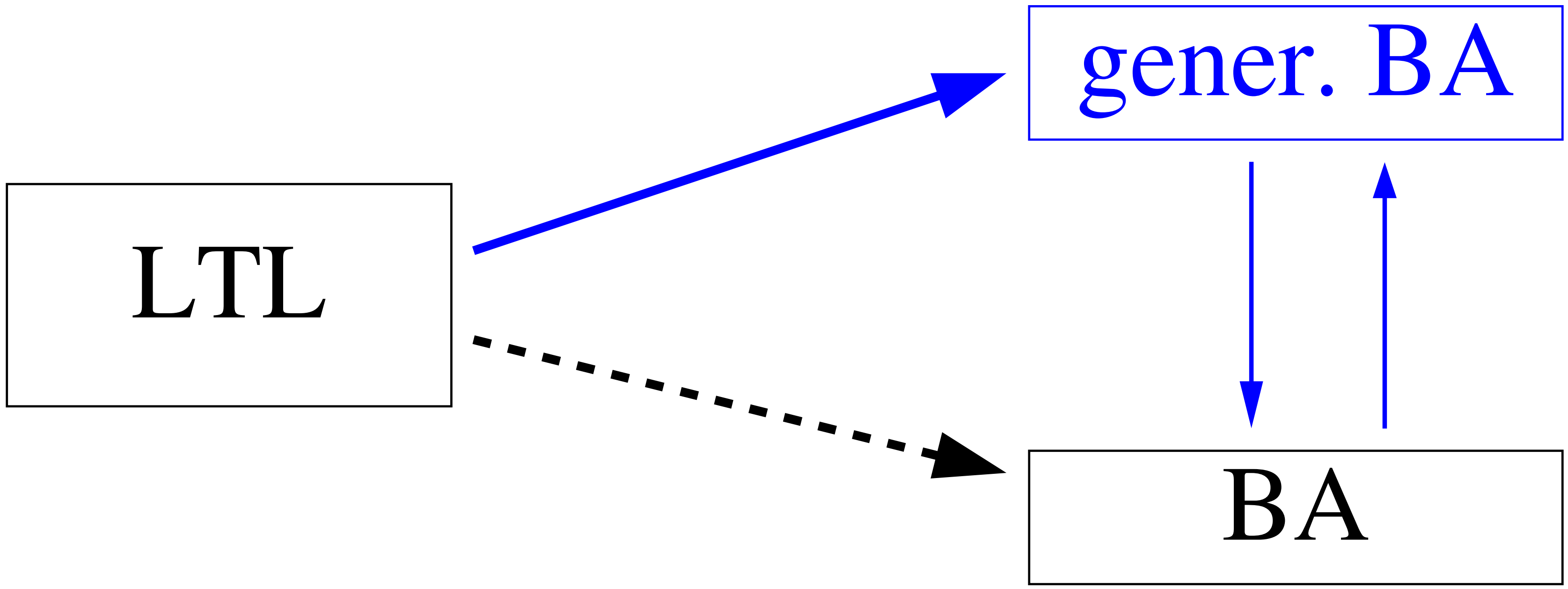
We desire to translate LTL formulae into Büchi automata.

Preview



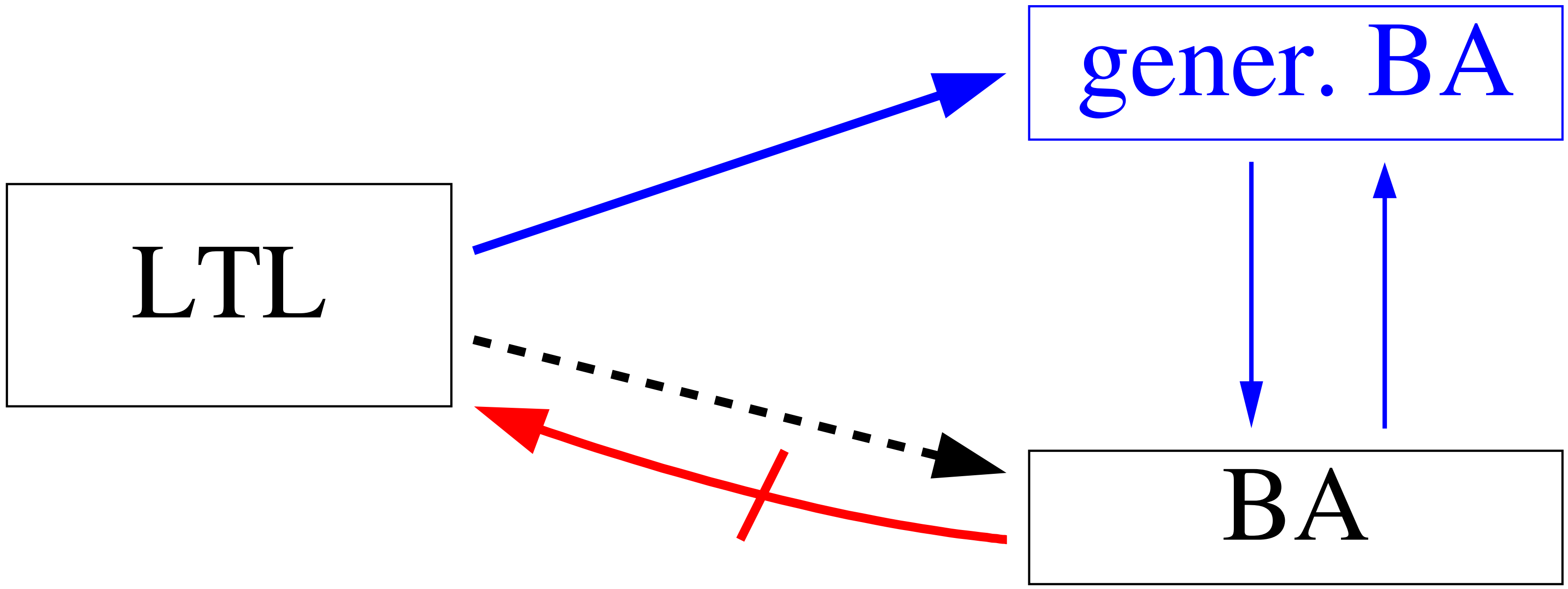
Detour: We translate them into so-called *generalized* Büchi automata (GBA).

Preview



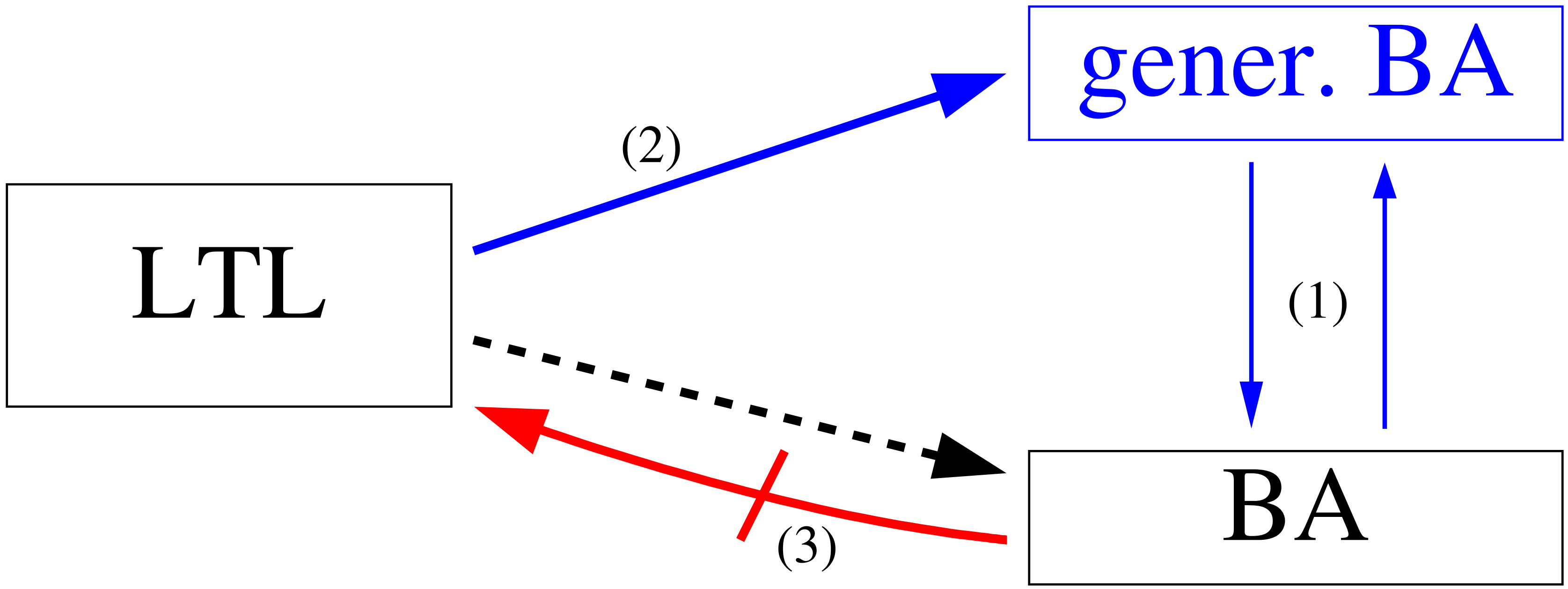
GBA accept the same class of languages as BA.

Preview



Translation from BA to LTL not possible in general.

Preview



We shall proceed in the order indicated above.

Generalized Büchi automata

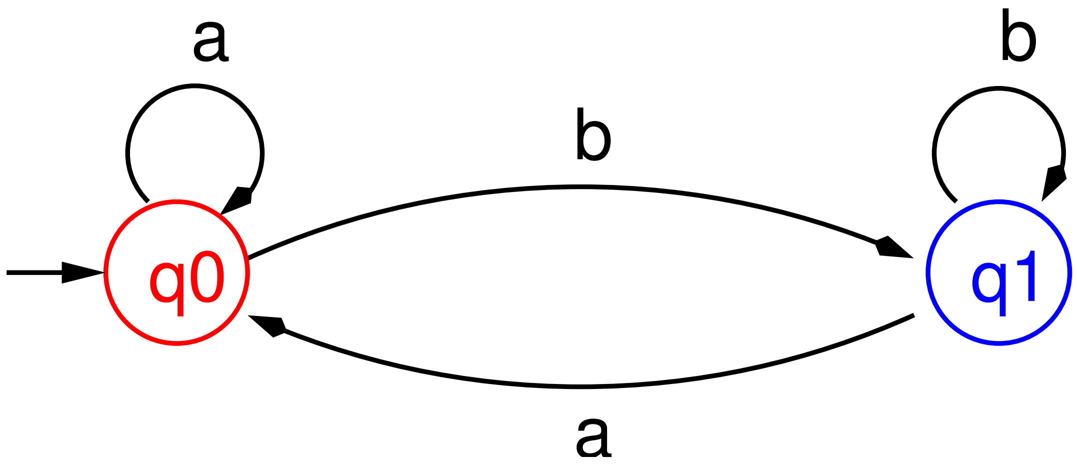
A generalized Büchi automaton (GBA) is a tuple $\mathcal{G} = (\Sigma, S, s_0, \Delta, \mathcal{F})$.

There is only one difference w.r.t. normal BA:

> The acceptance condition $\mathcal{F} \subseteq 2^S$ is a *set of sets of states*.
>
> E.g., let $\mathcal{F} = \{F_1, \ldots, F_n\}$. A run ρ of $\mathcal{G}$ is called accepting iff for every F_i ($i = 1, \ldots, n$), ρ visits infinitely many states of F_i.
>
> Put differently: many acceptance conditions at once.

GBA: Example

For the GBA shown below, let $\mathcal{F} = \{\ \{q_0\},\ \{q_1\}\ \}$.



Language of the automaton: “infinitely often *a and* infinitely often *b*”

Note: In general, the acceptance conditions need not be pairwise disjoint.

Advantage: GBA may be more succinct than BA.

Translations BA $\leftrightarrow$ GBA

GBA accept the same class of languages as BA.

I.e., for every BA there is a GBA accepting the same language, and vice versa.

Part 1 of the claim (BA $\rightarrow$ GBA):

Let $\mathcal{B} = (\Sigma, S, s_0, \Delta, F)$ be a BA.

Then $\mathcal{G} = (\Sigma, S, s_0, \Delta, \{F\})$ is a GBA with $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{B})$.

Part 2 of the claim (GBA $\rightarrow$ BA):

Let $\mathcal{G} = (\Sigma, S, s_0, \Delta, \{F_1, \ldots, F_n\})$ be a GBA.

We construct $\mathcal{B} = (\Sigma, S', s'_0, \Delta', F)$ as follows:

$S' = S \times \{1, \ldots, n\}$

$s'_0 = (s_0, 1)$

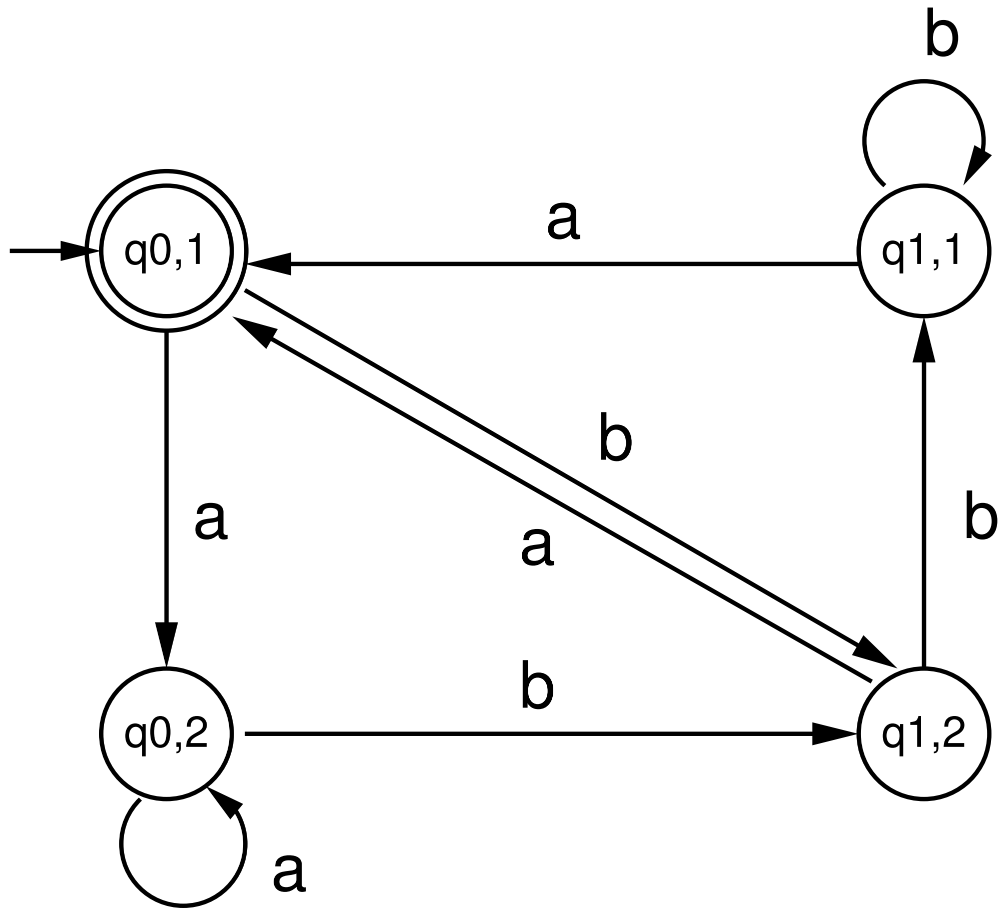
$F = F_1 \times \{1\}$

$((s, i), a, (s', k)) \in \Delta'$ iff $1 \leq i \leq n,\ (s, a, s') \in \Delta$
and $k = \begin{cases} i & \text{if } s \notin F_i \\ (i \bmod n) + 1 & \text{if } s \in F_i \end{cases}$

Then we have $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{G})$. (Idea: *n*-fold intersection)

GBA $\rightarrow$ BA: example

The BA corresponding to the previous GBA ("infinitely often *a and* infinitely often *b*") is as follows:



Remark: Multiple initial states

Our definitions of BA and GBA require exactly one initial state.

For the translation LTL $\rightarrow$ BA it will be convenient to use GBA with multiple initial states.

> Intended meaning: A word is regarded as accepted if it is accepted starting from *any* initial state.

Obviously, every (G)BA with multiple initial states can easily be converted into a (G)BA with just one initial state.

Part 5: LTL and Büchi automata

Overview

In this part, we shall solve the following problem:

Given an LTL formula ϕ over *AP*, we shall construct a GBA $\mathcal{G}$ (with multiple initial states) such that $\mathcal{L}(\mathcal{G}) = [\![\phi]\!]$.

($\mathcal{G}$ can then be converted to a normal BA.)

Remarks:

Analogous operation for regular languages: reg. expression $\longrightarrow$ NFA

The crucial difference: it is not possible to provide an LTL $\longrightarrow$ BA translation in modular fashion.

The automaton may have to check multiple subformulae *at the same time* (e.g.: $(\mathbf{G}\,\mathbf{F}\,p) \rightarrow (\mathbf{G}(q \rightarrow \mathbf{F}\,r))$ or $(p\,\mathbf{U}\,q)\,\mathbf{U}\,r$).

More remarks:

The construction shown in the following is comparatively simplistic.

It will produce rather suboptimal automata (size *always* exponential in $|\phi|$).

Obviously, this is quite inefficient, and not meant to be done by pen and paper, only as a “proof of concept”.

There are far better translation procedures but the underlying theory is rather beyond the scope of the course.

Interesting, on-going research area!

Structure of the construction

1. We first convert ϕ into a certain normal form.

2. States will be “responsible” for some set of subformulae.

3. The transition relation will ensure that “simple” subformulae such as p or $\mathbf{X}\,p$ are satisfied.

4. The acceptance condition will ensure that $\mathbf{U}$-subformulae are satisfied.

Negation normal form

Let *AP* be a set of atomic propositions. The set of NNF formulae over *AP* is inductively defined as follows:

- If $p \in AP$ then p and $\neg p$ are NNF formulae.
  (Remark: Negations occur *exclusively* in front of atomic propositions.)
- If ϕ_1 and ϕ_2 are NNF formulae then so are

$$\phi_1 \vee \phi_2, \quad \phi_1 \wedge \phi_2, \quad \mathbf{X}\, \phi_1, \quad \phi_1 \,\mathbf{U}\, \phi_2, \quad \phi_1 \,\mathbf{R}\, \phi_2.$$

Claim: For every LTL formula ϕ there is an equivalent NNF formula:

$$\begin{array}{rcl@{\qquad}rcl}
\neg(\phi_1 \,\mathbf{R}\, \phi_2) & \equiv & \neg\phi_1 \,\mathbf{U}\, \neg\phi_2 & \neg(\phi_1 \,\mathbf{U}\, \phi_2) & \equiv & \neg\phi_1 \,\mathbf{R}\, \neg\phi_2 \\
\neg(\phi_1 \wedge \phi_2) & \equiv & \neg\phi_1 \vee \neg\phi_2 & \neg(\phi_1 \vee \phi_2) & \equiv & \neg\phi_1 \wedge \neg\phi_2 \\
\neg\, \mathbf{X}\, \phi & \equiv & \mathbf{X}\, \neg\phi & \neg\neg\phi & \equiv & \phi
\end{array}$$

NNF: Example

Translation into an NNF formula:

$$
\begin{aligned}
\mathbf{G}(p \rightarrow \mathbf{F}\, q) &\equiv \neg\, \mathbf{F} \neg(p \rightarrow \mathbf{F}\, q) \\
&\equiv \neg(\mathbf{true}\ \mathbf{U} \neg(p \rightarrow \mathbf{F}\, q)) \\
&\equiv \neg\mathbf{true}\ \mathbf{R}\ (p \rightarrow \mathbf{F}\, q) \\
&\equiv \mathbf{false}\ \mathbf{R}\ (\neg p \vee \mathbf{F}\, q) \\
&\equiv \mathbf{false}\ \mathbf{R}\ (\neg p \vee (\mathbf{true}\ \mathbf{U}\ q))
\end{aligned}
$$

Remark: Because of this, we shall henceforth assume that the LTL formula in the translation procedure is given in NNF.

Subformulae

Let ϕ be an NNF formula. The set $Sub(\phi)$ is the smallest set satisfying:

$\phi \in Sub(\phi)$;

$\mathbf{true} \in Sub(\phi)$;

if $\phi_1 \in Sub(\phi)$ then $\neg\phi_1 \in Sub(\phi)$, and vice versa;

if $\mathbf{X}\,\phi_1 \in Sub(\phi)$ then $\phi_1 \in Sub(\phi)$;

if $\phi_1 \vee \phi_2 \in Sub(\phi)$ then $\phi_1, \phi_2 \in Sub(\phi)$;

if $\phi_1 \wedge \phi_2 \in Sub(\phi)$ then $\phi_1, \phi_2 \in Sub(\phi)$;

if $\phi_1 \,\mathbf{U}\, \phi_2 \in Sub(\phi)$ then $\phi_1, \phi_2 \in Sub(\phi)$;

if $\phi_1 \,\mathbf{R}\, \phi_2 \in Sub(\phi)$ then $\phi_1, \phi_2 \in Sub(\phi)$.

Note: We have $|Sub(\phi)| = \mathcal{O}(|\phi|)$ (one subformula per syntactic element).

Consistent sets

Recall item 2 of the construction:

> Every state will be labelled with a subset of $Sub(\phi)$.

Idea: A state labelled by set M will accept a sequence iff it satisfies every single subformula contained in M and violates every single subformula contained in $Sub(\phi) \setminus M$.

For this reason, we will a priori exclude some sets M which would obviously lead to empty languages.

The other states will be called consistent.

Definition: We call a set $M \subset Sub(\phi)$ *consistent* if it satisfies the following conditions:

$\mathbf{true} \in M$

if $\phi_1 \in Sub(\phi)$ then $\neg\phi_1 \in M$ gdw. $\phi_1 \notin M$;

if $\phi_1 \wedge \phi_2 \in Sub(\phi)$ then $\phi_1 \wedge \phi_2 \in M$ iff $\phi_1 \in M$ and $\phi_2 \in M$;

if $\phi_1 \vee \phi_2 \in Sub(\phi)$ then $\phi_1 \vee \phi_2 \in M$ iff $\phi_1 \in M$ or $\phi_2 \in M$.

By $CS(\phi)$ we denote the set of all consistent subsets of $Sub(\phi)$.

Translation (1)

Let ϕ be an NNF formula and $\mathcal{G} = (\Sigma, S, S_0, \Delta, \mathcal{F})$ be a GBA such that:

$\Sigma = 2^{AP}$
(i.e. the valuations over *AP*)

$S = CS(\phi)$
(i.e. every state is a consistent set)

$S_0 = \{ M \in S \mid \phi \in M \}$
(i.e. the initial states admit sequences satisfying ϕ)

Δ and $\mathcal{F}$: see next slide

Translation (2)

Transitions: $(M, \sigma, M') \in \Delta$ iff $\sigma = M \cap AP$ and:

– if $\mathbf{X}\,\phi_1 \in Sub(\phi)$ then $\mathbf{X}\,\phi_1 \in M$ iff $\phi_1 \in M'$;

– if $\phi_1\ \mathbf{U}\ \phi_2 \in Sub(\phi)$ then $\phi_1\ \mathbf{U}\ \phi_2 \in M$
  iff $\phi_2 \in M$ or ($\phi_1 \in M$ and $\phi_1\ \mathbf{U}\ \phi_2 \in M'$);

– if $\phi_1\ \mathbf{R}\ \phi_2 \in Sub(\phi)$ then $\phi_1\ \mathbf{R}\ \phi_2 \in M$
  iff $\phi_1 \wedge \phi_2 \in M$ or ($\phi_2 \in M$ and $\phi_1\ \mathbf{R}\ \phi_2 \in M'$).

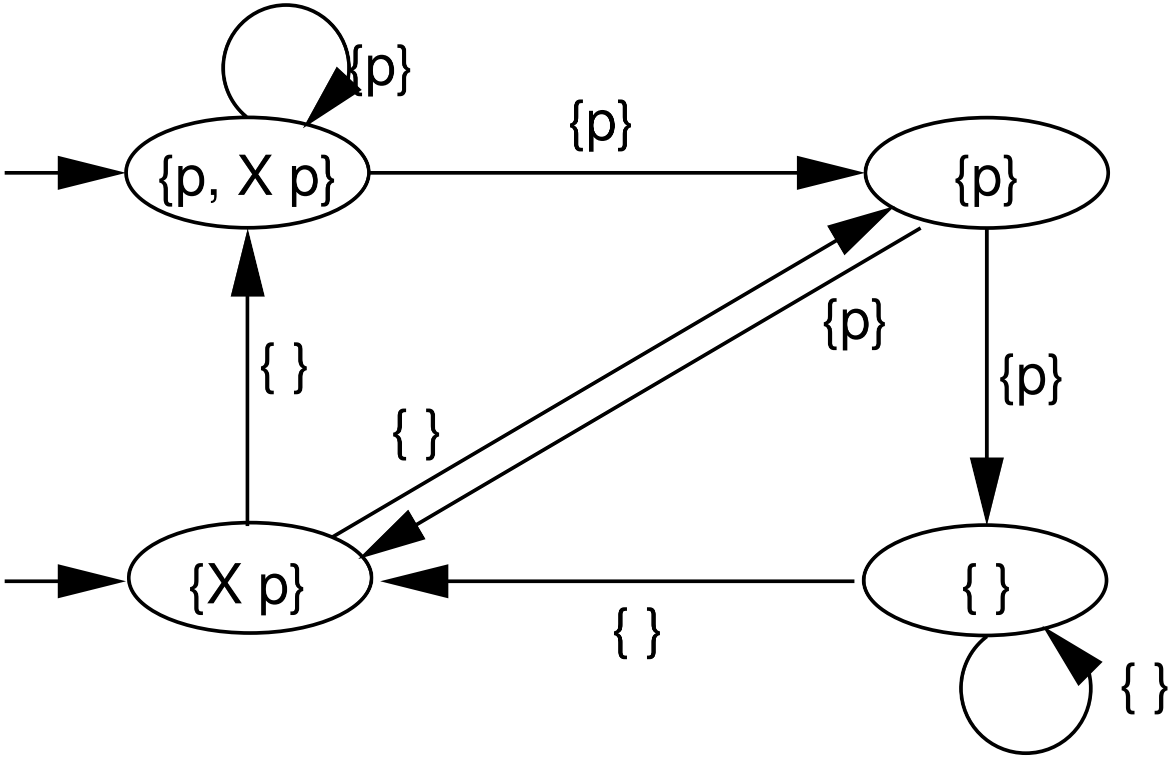
Acceptance condition:

$\mathcal{F}$ contains a set F_ψ for every subformula ψ of the form $\phi_1\ \mathbf{U}\ \phi_2$, where

$$F_\psi = \{\, M \in CS(\phi) \mid \phi_2 \in M \text{ or } \neg(\phi_1\ \mathbf{U}\ \phi_2) \in M \,\}.$$

Translation: Example 1

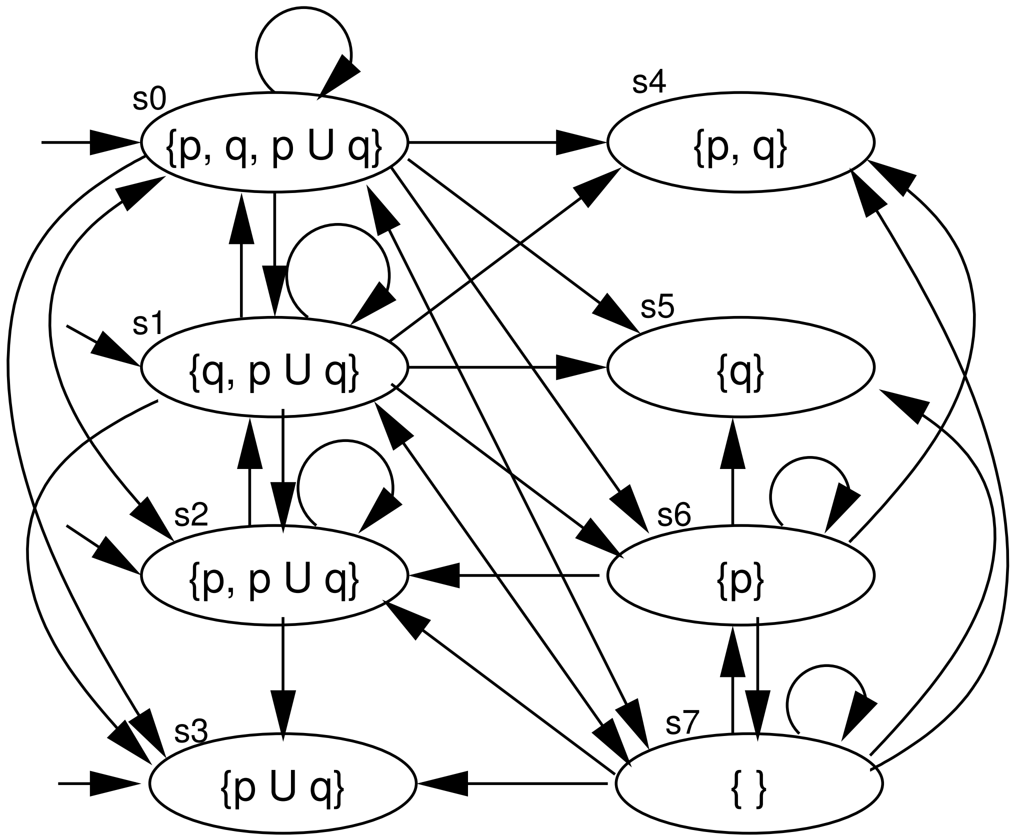
$\phi = \mathbf{X}\, p$



This GBA has got two initial states and the acceptance condition $\mathcal{F} = \emptyset$, i.e. every infinite run is accepting. (Negated Formulas omitted from state labels.)

Translation: Example 2

$\phi \equiv p \, \mathbf{U} \, q$



GBA with $\mathcal{F} = \{\{s_0, s_1, s_4, s_5, s_6, s_7\}\}$, transition labels also omitted.

Proof of correctness

We want to prove the following:

$$\sigma \in \mathcal{L}(\mathcal{G}) \quad \text{gdw.} \quad \sigma \in [\![\phi]\!]$$

To this aim, we shall prove the following stronger property:

Let α be a sequence of consistent sets (i.e., states of $\mathcal{G}$)
and let σ be a sequence of valuations over AP.

α is an accepting run of $\mathcal{G}$ over σ
iff $\sigma^i \in [\![\psi]\!]$ for all $i \geq 0$ and $\psi \in \alpha(i)$.

The desired proof then follows from the choice of initial states.

Correctness (2)

Remark: By construction, we have $\sigma(i) = \alpha(i) \cap AP$ for all $i \geq 0$.

Proof via structural induction over ψ:

for $\psi = p$ and $\psi = \neg p$ if $p \in AP$:
obvious since $\sigma^i \in [\![p]\!]$ iff $p \in \sigma(i)$ iff $p \in \alpha(i)$.

for $\psi_1 \vee \psi_2$ and $\psi_1 \wedge \psi_2$: follows from consistency of $\alpha(i)$ and from the induction hypothesis for ψ_1 and ψ_2, resp.

for $\mathbf{X}\,\psi_1$: follows from the construction of Δ and induction hypothesis for ψ_1.

Correctness (3)

for $\psi = \psi_1 \,\mathbf{R}\, \psi_2$:

Follows from the construction of Δ, the recursion equation for $\mathbf{R}$ and the induction hypothesis.

for $\psi = \psi_1 \,\mathbf{U}\, \psi_2$:

Analogous to $\mathbf{R}$, but additionally we must ensure that $\psi_2 \in \alpha(k)$ for some $k \geq i$. Assume that this is not the case, then we have $\psi_1 \,\mathbf{U}\, \psi_2 \in \alpha(k)$ for all $k \geq i$. However, none of these states is in F_ψ, therefore α cannot be accepting, which is a contradiction.

Complexity of the translation

The translation procedure produces an automaton of size $\mathcal{O}(2^{|\phi|})$, for a formula ϕ.

Question: Is there a better translation procedure?

Answer 1: No (not in general). There exist formulae for which any Büchi automaton has necessarily exponential size.

Example: The following LTL formula over $\{p_0, \ldots, p_{n-1}\}$ simulates an n-bit counter.

$$\mathbf{G}(p_0 \not\leftrightarrow \mathbf{X}\, p_0) \wedge \bigwedge_{i=1}^{n-1} \mathbf{G}\left(\left(p_i \not\leftrightarrow \mathbf{X}\, p_i\right) \leftrightarrow \left(p_{i-1} \wedge \neg \mathbf{X}\, p_{i-1}\right)\right)$$

The formula has size $\mathcal{O}(n)$. Obviously, any automaton for this formula must have at least 2^n states.