Tecniche di Specifica e di Verifica

Automata-based
LTL Model-Checking

Finite state automata

A finite state automaton is a tuple A = (S,*S*,*S*$_0$,*R*,*F*)

- **S: set of input symbols**
- ***S*: set of *states* -- S$_0$: set of *initial states* (S$_0$ Í *S*)**
- ***R*:*S* ´ S ® 2^{S} : the *transition relation*.**
- ***F*: *set of accepting states* (F Í *S*)**
- **A *run r* on *w*=*a*$_1$,…,*a*$_n$ is a sequence *s*$_0$,…,*s*$_n$ such that *s*$_0$ Î *S*$_0$ and *s*$_{i+1}$ Î R(s$_i$,*a*$_i$) for 0£i£n.**
- **A *run r* is *accepting* if *s*$_n$ Î *F*, while a word w is *accepted* by *A* if there is an accepting run of A on w.**
- **The *language ℒ(A) accepted* by A is the set of finite words accepted by A.**

Finite state automata: union

Given automata A_1 and A_2, there is an automaton A accepting $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$

$A = (\Sigma, S, S_0, R, F)$ is an automaton which just runs non-deterministically either A_1 or A_2 on the input word.

$$S = S_1 \cup S_2$$

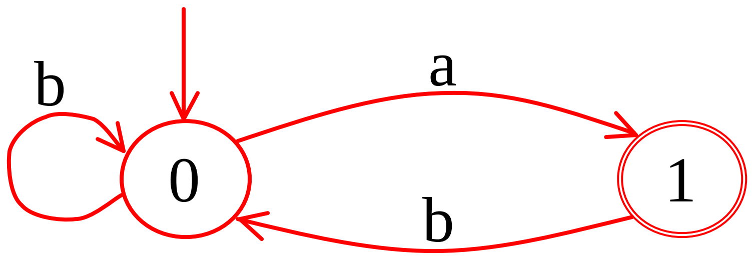
$$F = F_1 \cup F_2$$

$$S_0 = S_{01} \cup S_{02}$$

$$R(s,a) = \begin{cases} R_1(s,a) & \text{if } s \in S_1 \\ R_2(s,a) & \text{if } s \in S_2 \end{cases}$$

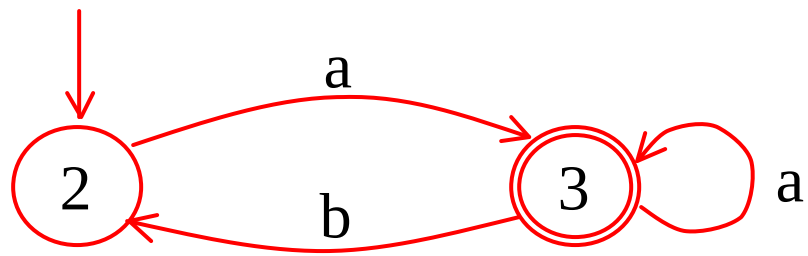
Finite state automata: union

A_1

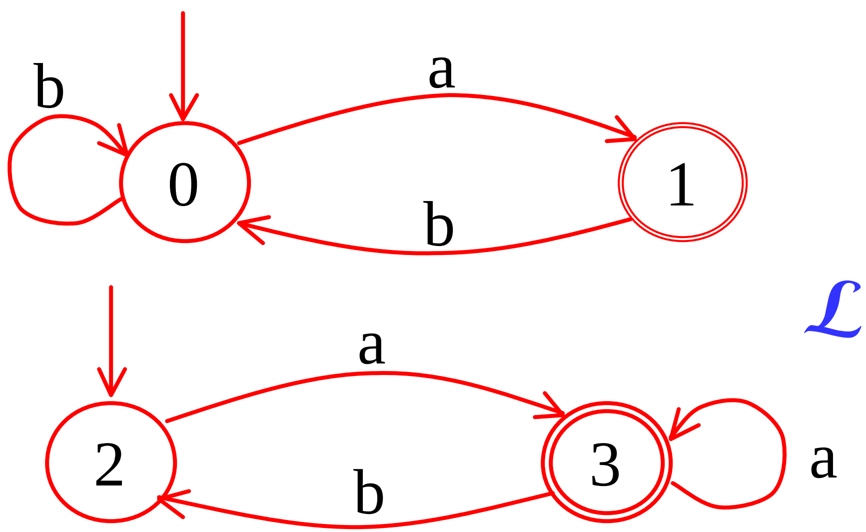


$\mathcal{L}(A_1)=b^*(ab)^*a$

A_2



$\mathcal{L}(A_2)=a(a*ba)^*$



A_1 È A_2

$\mathcal{L}(A)=\mathcal{L}(A_1)$ È $\mathcal{L}(A_2)$

Finite state automata: intersection

Given automata A_1 and A_2, there is an automaton A accepting $\mathcal{L}(A) = \mathcal{L}(A_1)$ Ç $\mathcal{L}(A_2)$

$A = (S,S,S_0,R,F)$ runs simultaneously both automata A_1 and A_2 on the input word.

$S = S_1$ ´ S_2

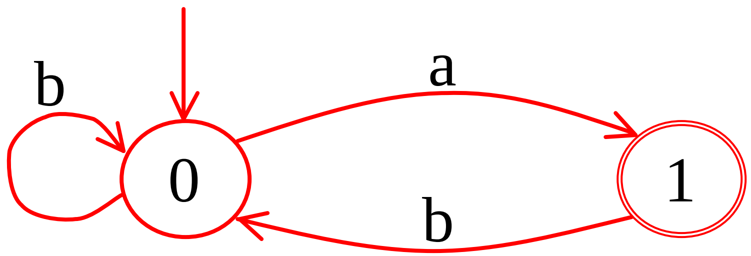
$F = F_1$ ´ F_2

$S_0 = S_{01}$ ´ S_{02}

$R((s,t),a) = R_1(s,a)$ ´ $R_2(t,a)$

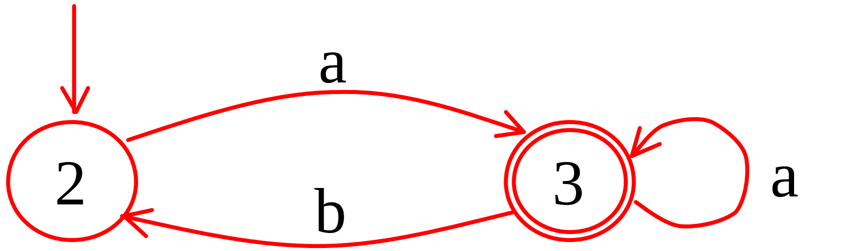
Finite state automata: intersection

A_1



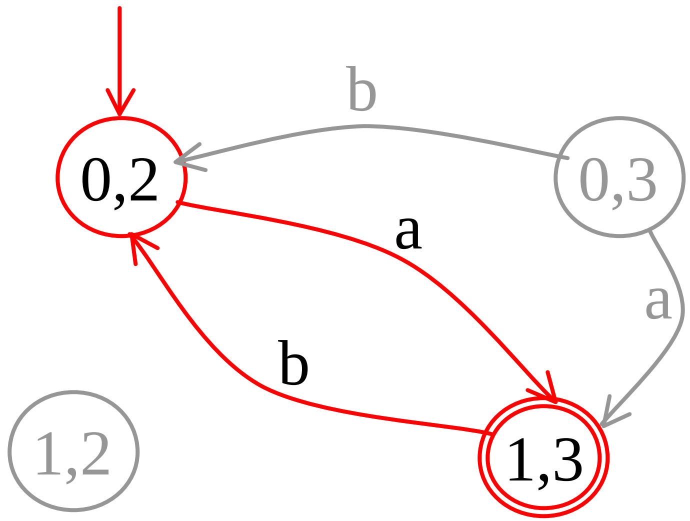
$\mathcal{L}(A_1)=b^*(ab)^*a$

A_2



$\mathcal{L}(A_2)=a(a^*ba)^*$

A_1 Ç A_2

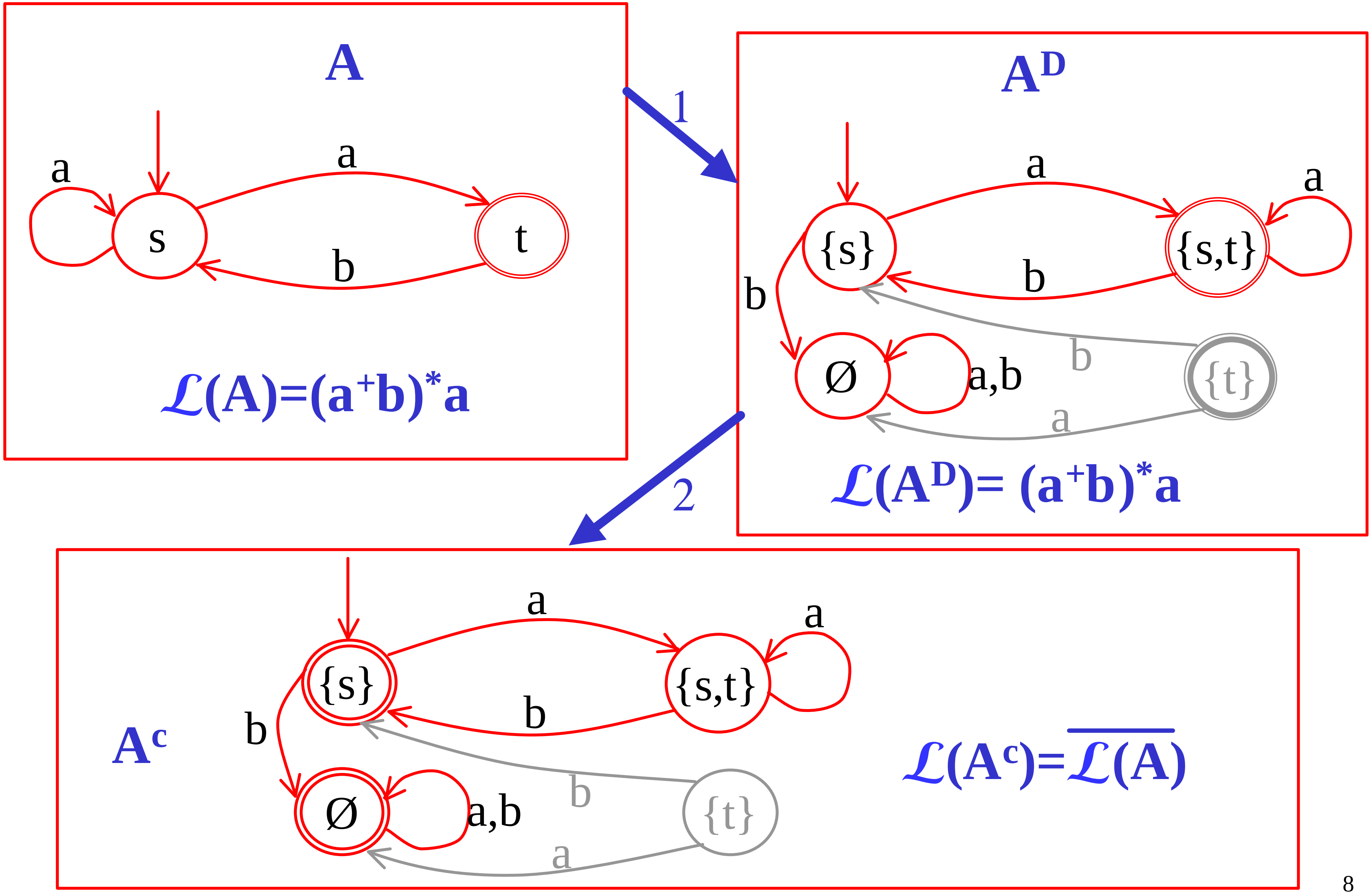


$\mathcal{L}(A)=L(A_1)$ Ç $\mathcal{L}(A_2)$

Finite state automata: complementation

- **If the automaton is deterministic, then it just suffices to set $F^c = S - F$.**
- **This doesn't work, though, for *non-deterministic automata*.**
- **Solution:**
  1. ***Determinize* the automaton using the subset construction.**
  2. ***Complement* the resulting deterministic automaton**
- **The complexity of this process is *exponential* in the size of the original automaton.**
- **The number of states of the final automaton is $2^{|S|}$, in the *worst case*.**

Finite state automata: complementation



Büchi automata (BA)

A Büchi automaton is a tuple A = *(S,S,S$_0$,R,F)*

- **S: set of input symbols**
- ***S*: set of *states* -- S$_0$: set of *initial* states (S$_0$ Í *S*)**
- ***R:S* ´ S ® 2^{S} : the *transition relation*.**
- ***F*: *set of accepting states* (F Í *S*)**
- **A *run r* on *w=a$_1$,a$_2$,…* is an infinite sequence *s$_0$,s$_1$,…* such that *s$_0$* Î *S$_0$* and *s$_{i+1}$* Î R(s$_i$,*a$_i$*) for i ³ 0.**
- **A *run r* is *accepting* if some *accepting state in F* occurs in *r infinitely often*.**
- **A word w is *accepted* by *A* if there is an accepting run of A on w, and the *language* $\mathcal{L}_{w}$*(A) accepted* by A is the set of (infinite) w-words accepted by A.**

Büchi automata (BA)

A Büchi automaton is a tuple $A = (\Sigma, S, S_0, R, F)$

- **A *run r* on $w = a_1, a_2, \ldots$ is an infinite sequence $s_0, s_1, \ldots$ such that $s_0 \in S_0$ and $s_{i+1} \in R(s_i, a_i)$ for $i \geq 0$.**
- **Let $Lim(r) = \{\, s \mid s = s_i$ for infinitely many $i \,\}$**
- **A *run r* is *accepting* if**

$$Lim(r) \cap F \neq \emptyset$$

- **A word w is *accepted* by A if there is an accepting run of A on w.**
- **The *language* $\mathcal{L}_\omega(A)$ *accepted* by A is the set of (infinite) ω-words accepted by A.**

Büchi automata: union

Given Büchi automata A_1 and A_2, there is an Büchi automaton A accepting $\mathcal{L}_\omega(A) = \mathcal{L}_\omega(A_1) \cup \mathcal{L}_\omega(A_2)$.

The *construction* is the same as for *ordinary automata*.

$A = (\Sigma, S, S_0, R, F)$ is an automaton which just runs non-deterministically either A_1 or A_2 on the input word.

$$S = S_1 \cup S_2$$

$$F = F_1 \cup F_2$$

$$S_0 = S_{01} \cup S_{02}$$

$$R(s,a) = \begin{cases} R_1(s,a) & \text{if } s \in S_1 \\ R_2(s,a) & \text{if } s \in S_2 \end{cases}$$

Büchi automata: intersection

- **The intersection construction for automata does not work for Büchi automata.**
- **Instead, the intersection for Büchi automata can be defined as follows:**

$A=(\Sigma,S,S_0,R,F)$ **intuitively runs simultaneously both automata** $A_1=(\Sigma,S_1,S_{01},R_1,F_1)$ **and** $A_2=(\Sigma,S_2,S_{02},R_2,F_2)$ **on the input word.**

$$S = S_1 \times S_2 \times \{1,2\}$$

$$F = F_1 \times S_2 \times \{1\}$$

$$S_0 = S_{01} \times S_{02} \times \{1\}$$

$$R((s,t,i),a) = \begin{cases} (s',t',2) & \text{if } s' \in R_1(s,a),\ t' \in R_2(t,a),\ s \in F_1 \text{ and } i=1 \\ (s',t',1) & \text{if } s' \in R_1(s,a),\ t' \in R_2(s,a),\ t \in F_2 \text{ and } i=2 \\ (s',t',i) & \text{if } s' \in R_1(s,a),\ t' \in R_1(t,a) \end{cases}$$

Büchi automata: intersection

$A = (\Sigma,S,S_0,R,F)$ runs simultaneously both automata A_1 and A_2 on the input word.

$$S = S_1 \times S_2 \times \{1,2\}$$

$$F = F_1 \times S_2 \times \{1\}$$

$$S_0 = S_{01} \times S_{02} \times \{1\}$$

$$R((s,t,i),a) = \begin{cases} (s',t',2) & \text{if } s' \in R_1(s,a),\ t' \in R_2(t,a),\ s \in F_1 \text{ and } i=1 \\ (s',t',1) & \text{if } s' \in R_1(s,a),\ t' \in R_2(t,a),\ t \in F_2 \text{ and } i=2 \\ (s',t',i) & \text{if } s' \in R_1(s,a),\ t' \in R_1(t,a) \end{cases}$$

The automaton remembers *2 tracks*, one for each automaton, and *points* to one of the tracks. As soon as it goes through an accepting state on the current track, it changes track.

The accepting condition and the transition relation ensure that this change of track must happens infinitely often.

Büchi automata: intersection

A = (S,S,S_0,R,F) runs simultaneously both automata A_1 and A_2 on the input word.

$$S = S_1 ´\ S_2 ´\ \{1,2\}$$

$$F = F_1 ´\ S_2 ´\ \{1\}$$

$$S_0 = S_{01} ´\ S_{02} ´\ \{1\}$$

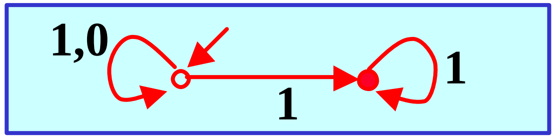
$$R((s,t,i),a) = \begin{cases} (s',t',2) & \text{if } s' \text{ Î } R_1(s,a),\ t' \text{ Î } R_2(t,a),\ s \text{ Î } F_1 \text{ and } i=1 \\ (s',t',1) & \text{if } s' \text{ Î } R_1(s,a),\ t' \text{ Î } R_2(t,a),\ t \text{ Î } F_2 \text{ and } i=2 \\ (s',t',i) & \text{if } s' \text{ Î } R_1(s,a),\ t' \text{ Î } R_1(t,a) \end{cases}$$

As soon as it visits an accepting state in *track 1*, it switches to *track 2* and then to *track 1* again but only after visiting an accepting state in the *track 2*.

Therefore, to visit *infinitely often* a state in F (F_1), the automaton must also visit *infinitely often* some state of F_2.

Büchi automata: complementation

It's a complicated construction -- the standard subset construction for *determinizing automata doesn't work* as *non-deterministic automata* are *more powerful* than *deterministic ones* (e.g. $\mathcal{L}_\omega=(0+1)^*1^\omega$)



Solution (resorts to another kind of automaton):

- **Transform the (non-deterministic) Büchi automaton into a (non-deterministic) *Rabin automaton* (a more general kind of ω-automaton).**
- **Determinize and then complement the Rabin automaton.**
- **Transform the Rabin automaton into a Büchi automaton.**

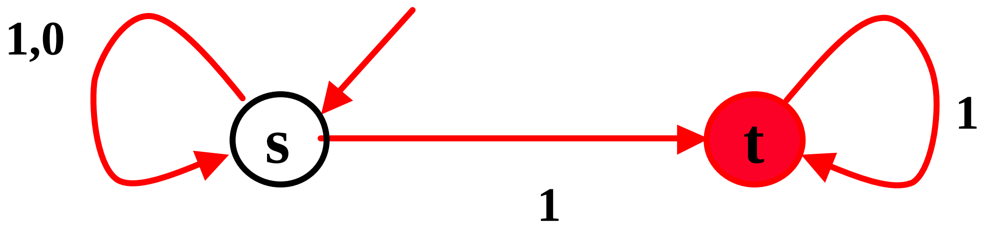
- **Therefore, also *Büchi automata are closed under complementation*.**

Rabin automata

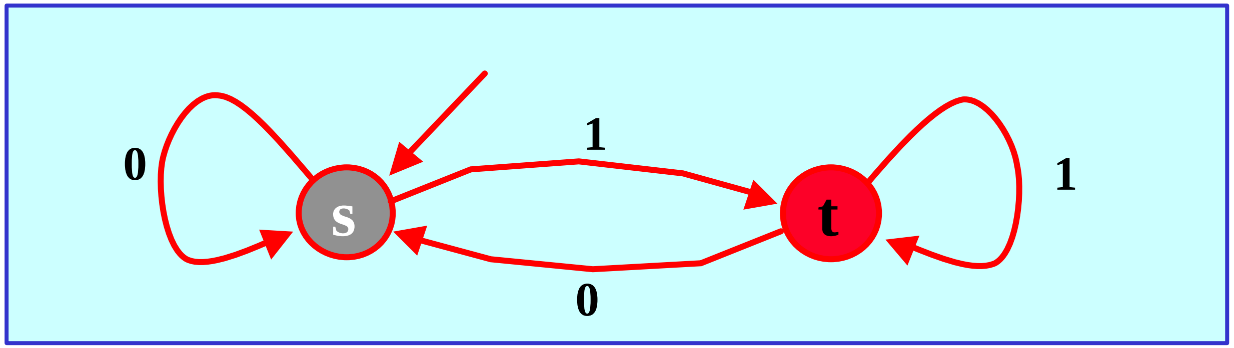
- **A Rabin automaton is like a Büchi automaton, except that the accepting condition is defined differently.**
- ***A = (S,S,S_0,R,F)*, where *F=((G_1,B_1),…,(G_m,B_m))*.**
- **and the acceptance condition for a run *r = s_0,s_1,…* is as follows: for some *i***
  - ***Lim(r)* Ç *G_i* ¹ Æ and**
  - ***Lim(r)* Ç *B_i* = Æ**

in other words, there is a pair *(G_i,B_i)* such that the "*good*" set (G_i) is visited *infinitely often*, while the "*bad*" set (B_i) is visited only *finitely often*.

Rabin versus Büchi automata



The Büchi automaton
fot $\mathcal{L}_w = (0+1)^* 1^w$



The **Rabin automaton**
fot $\mathcal{L}_w = (0+1)^* 1^w$

The Rabin automaton has $F=((\{t\},\{s\}))$

Note that the Rabin automaton is ***deterministic***.

Language emptiness for Büchi automata

The ***emptiness problem for Büchi automata*** is the problem of ***deciding*** whether the language accepted by a Büchi automaton **A** is empty, i.e. if **ℒ(A) = Æ**.

Theorem: The ***emptiness problem for Büchi automata*** is ***decidable in linear time***, i.e. in time **O(|A|)**.

Fact: **ℒ(A) = Æ** ***iff*** in the Büchi automaton there is ***no reachable cycle*** **A** ***containing a state in F***.

Language emptiness for Büchi automata

In other words, ***$\mathcal{L}$(A)*** ¹ Æ ***iff*** there is a ***cycle*** containing an ***accepting state***, which is also ***reachable from some initial state*** of the automaton.

We need to find whether there is such a reachable cycle

We could simply compute the ***SCCs*** of **A** using the standard ***DFS*** algorithm, and check if there exists a reachable (***nontrivial***) ***SCC*** containing a state in ***F***.

But this is usually ***too inefficient*** in practice. We will therefore use a ***more efficient nested DFS*** (more efficient in the ***average-case***).

Efficient language emptiness for BA

Input: A

Initialize: $Stack_1$:=Æ, $Stack_2$:= Æ
$Table_1$:= Æ, $Table_2$:= Æ

Algorithm Main()

    foreach s Î Init
        if s Ï $Table_1$ then
            DFS1(s);
    output("empty");
    return;

Algorithm DFS1(s)

    push(s,$Stack_1$);
    hash(s,$Table_1$);
    foreach t Î Succ(s)
        if t Ï $Table_1$ then
            DFS1(t);
    if s Î F then
        DFS2(s);
    pop($Stack_1$);

Algorithm DFS2(s)

    push(s,$Stack_2$);
    hash(s,$Table_2$) ;
    foreach t Î Succ (s) do
        if t Ï $Table_2$ then
            DFS2(t)
        else if t is on $Stack_1$
            output("not empty");
            output($Stack_1$,$Stack_2$,t);
            return;
    pop($Stack_2$);

<u>Note</u>: upon finding a bad cycle, $Stack_1$+$Stack_2$+t, determines a counterexample: a bad cycle reached from an init state.

Generalized Büchi automata (GBA)

Generalized Büchi automaton: **A = (*S ,S,S*$_0$*,R,(F*$_1$*,...,F*$_m$*))***

- **A *run r* on *w=a*$_1$*,a*$_2$*,…* is an infinite sequence *s*$_0$*,s*$_1$*,…* such that *s*$_0$ Î *S*$_0$ and *s*$_{i+1}$ Î R(s$_i$,*a*$_i$) for i ³ 0.**
- **Let *Lim*(*r*) = { *s* | *s* = *s*$_i$ for infinitely many *i* }**
- **A *run r* is *accepting* if for each *1* £ *i* £ *m***

***Lim*(*r*) Ç *F*$_i$ ¹ Æ**

Any *Generalized Büchi automaton* can be easily transformed into a *Büchi automaton* as follows:

ℒ(S ,S,S$_0$*,R,(F*$_1$*,...,F*$_m$*)) =* Ç$_{i\,Î\,\{1,\dots,m\}}$ *ℒ(S ,S,S*$_0$*,R,F*$_i$*)*

This transformation is *not very efficient*, though.

From GBA to BA efficiently

Generalized Büchi automaton: $A = (S, S, S_0, R, (F_1, \ldots, F_m))$

A *Generalized Büchi automaton* can be *efficiently* transformed into a *Büchi automaton* as follows:

$$S' = S \text{ ´ } \{1,\ldots,m\}$$

$$F' = F \text{ ´ } \{i\} \text{ for some } 1 \text{ £ } i \text{ £ } m$$

$$S'_0 = S_0 \text{ ´ } \{i\} \text{ for some } 1 \text{ £ } i \text{ £ } m$$

$$R((s,i),a) = \begin{cases} (s', (i \bmod m)+1) & \text{if } s' \text{ Î } R(s,a) \text{ and } s \text{ Î } F_i \\ (s',i) & \text{if } s' \text{ Î } R(s,a) \text{ and } s \text{ Ï } F_i \end{cases}$$

Notice that the transformation above expands the automaton size by a factor of *m* (compare with *Büchi Intersection*).

LTL-semantics and Büchi automata

- **We can interpret a formula y as expressing a property of w-words, i.e., an w-language L(y) Í S_{AP}^{w}.**
- **For w-word s = s_0, s_1, s_2,..... Î S_{AP}^{w}, let s^{i} = s_i, s_{i+1}, s_{i+2}.... be the suffix of s starting at position *i*. We defined the "*satisfies*" relation, ⊨ , inductively:**
  - s ⊨ p_j iff p_j Î s_0 (for any p_j Î P).
  - s ⊨ Øy iff not s ⊨ y.
  - s ⊨ y_1 Ú y_2 iff s ⊨ y_1 or s ⊨ y_2.
  - s ⊨ Xy iff s^1 ⊨ y.
  - s ⊨ y_1 U y_2 iff \$ *i* ³ 0 such that s^i ⊨ y_2,
    and "j, 0£ j< i, s^j ⊨ y_1.
- **We can then define the language ℒ(y) = { s | s ⊨ y }.**

Relation with Kripke structures

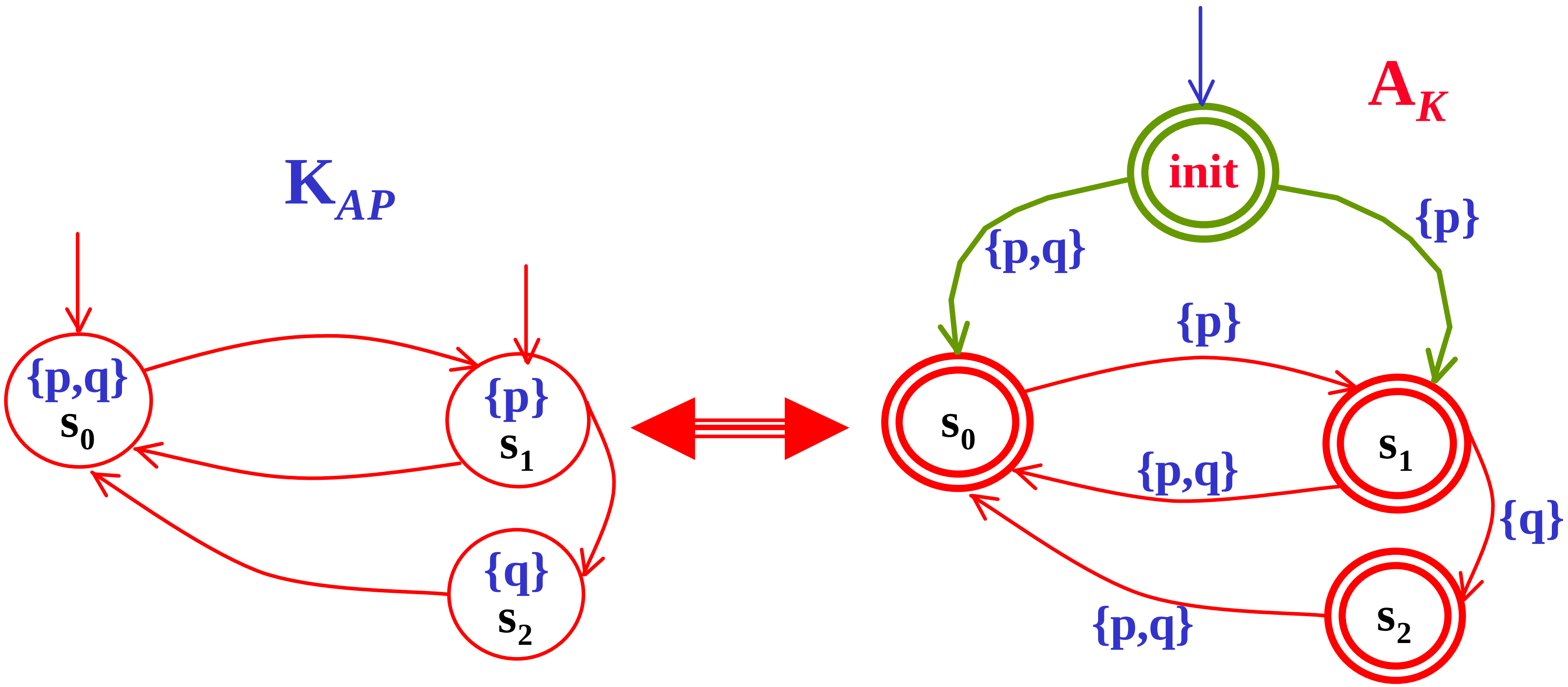
We extend our definition of "*satisfies*" to transition systems, or *Kripke structures*, as follows:

- $\mathbf{K}_{AP} \vDash \psi$ **iff** ***for all*** **computations (runs)** π **of** $\mathbf{K}_{AP}$**,** $\mathcal{L}(\pi) \vDash \psi$ **, or in other words, iff**

$$\mathcal{L}(\mathbf{K}_{AP}) \subseteq \mathcal{L}(\psi).$$

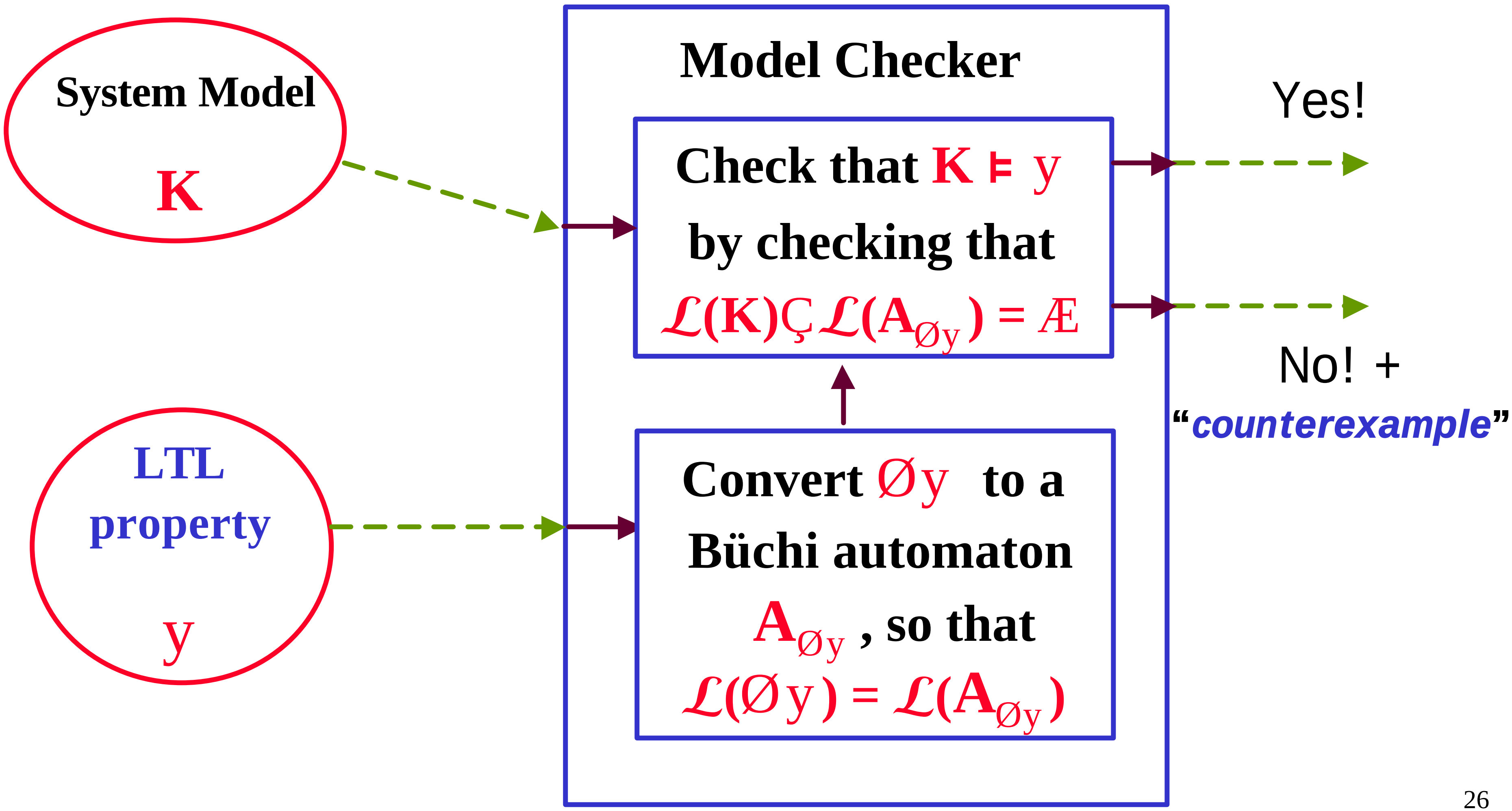
Relation with Kripke structures

We could transform any Kripke structure into a Büchi automaton as follows:



where *every state is accepting*!

LTL Model Checking



LTL Model Checking: explanation

M ⊧ y Û $\mathcal{L}(\mathbf{K}_{AP})$ Í $\mathcal{L}$(y)

Û $\mathcal{L}(\mathbf{K}_{AP})$ Ç $(\mathrm{S}_{AP}{}^{\mathrm{w}}$ \ $\mathcal{L}$(y)) = Æ

Û $\mathcal{L}(\mathbf{K}_{AP})$ Ç $\mathcal{L}$(Øy) = Æ

Û $\mathcal{L}(\mathbf{K}_{AP})$ Ç $\mathcal{L}(\mathbf{A}_{\text{Øy}})$ = Æ

Û $\mathcal{L}(\mathbf{K}_{AP}$ Ç $\mathbf{A}_{\text{Øy}})$ = Æ

The algorithmic tasks to perform

We have reduced LTL *model checking* to two tasks:

1 **Convert an LTL formula j (i.e. Øy) into a Büchi automaton A_j , such that $\mathcal{L}(j) = \mathcal{L}(A_j)$.**

- **Can we do this in general? …. Yes!!!……**

2 **Check whether $K_{AP} \models y$, by checking whether the intersection of languages $\mathcal{L}(K_{AP})$ Ç $\mathcal{L}(A_{Øy})$ is empty.**

- **It is actually unwise to first construct all of K_{AP}, because K_{AP} can be far too big (*state explosion*).**
- **Instead, it is possible perform the check by *constructing* states of K_{AP} only *as needed*.**

LTL to BA translation

- First, let's put LTL formulas φ in ***normal form*** where:
  - Ø 's have been "**pushed in**", applying only to propositions.
  - the only propositional operators are Ø, Ù,Ú.
  - the only temporal operators are **X**, **U** and its dual **R**.
- In order to do that we use the following rules:
  - p ® q º Ø p Ú q ; p « q º (Ø p Ú q) Ù (Ø q Ú p)
  - Ø(p Ú q) º Ø p Ù Ø q ; Ø(p Ù q) º Ø p Ú Ø q ; Ø Ø p º p
  - Ø(p **U** q) º (Ø p) **R** (Ø q) ; Ø (p **R** q) º (Ø p) **U** (Ø q)
  - F p º ⊤ **U** p ; G p º ⊥ **R** p ; Ø X p º X Ø p

LTL to BA translation

- First, let’s put LTL formulas φ in ***normal form***
  - Ø ‘s have been “**pushed in**”, applying only to propositions.
- We use the following rules:
  - p ® q º Ø p Ú q ; p « q º (Ø p Ú q) Ù (Ø q Ú p)
  - Ø(p Ú q) º Ø p Ù Ø q ; Ø(p Ù q) º Ø p Ú Ø q ; Ø Ø p º p
  - Ø (p **U** q) º (Ø p) **R** (Ø q) ; Ø (p **R** q) º (Ø p) **U** (Ø q)
  - F p º T **U** p ; G p º ⊥ **R** p ; Ø X p º X Ø p

Examples:

((p U q) ® F r) º Ø(p U q) Ú F r º Ø(p U q) Ú (T U r) º

º (Ø p R Ø q) Ú (T U r)

GF p ® F r º (⊥ **R** (Fp)) ® (T U p) º (⊥ **R** (T U p)) ® (T U r) º

º Ø (⊥ **R** (T U p)) Ú (T U r) º (T U Ø (T U p)) Ú (T U r) º

º (T U (⊥ R Ø p)) Ú (T U r)

LTL to BA translation

- States of **A**$_j$ will be ***sets of subformulas*** of j , thus if we have j = **p**$_1$UØ**p$_2$**, a state is given by G Í **{p$_1$,Øp$_2$,p$_1$UØp$_2$}**.
- Consider a word s = s$_{\mathbf{0}}$, s$_{\mathbf{1}}$, s$_{\mathbf{2}}$,... Î S$_{\mathbf{AP}}^{w}$ such that s ⊧ φ, where, e.g., j = y$_1$**U**y$_{\mathbf{2}}$.
- Mark each position i with the set of subformulas Γ$_i$ of φ that hold true there:

G$_{\mathbf{0}}$ G$_{\mathbf{1}}$ G$_{\mathbf{2}}$ ....

s$_{\mathbf{0}}$ s$_{\mathbf{1}}$ s$_{\mathbf{2}}$ ....

- Clearly, j Î G$_{\mathbf{0}}$. But then, by ***consistency***, either:
  - y$_1$ Î G$_{\mathbf{0}}$ and j Î G$_{\mathbf{1}}$, or
  - y$_2$ Î G$_{\mathbf{0}}$.
- The consistency rules dictate our states and transitions.

LTL to BA translation

Let **sub(**j **)** denote the set of subformulas of j .

We define **A**$_{j}$ **= (Q,** S **, R, L, Init, F)** as follows.

First, the ***set of states*** of **A**$_{j}$ is defined as follows:

- **Q = {**G Í **sub(**j **)** | s.t. G is ***locally consistent*** **}**.
  - For G to be ***locally consistent*** we should, e.g., have:
    - ^ Ï G
    - if y Ú g Î G, then y Î G or g Î G.
    - if y Ù g Î G, then y Î G and g Î G.
    - if **p**$_{i}$ Î G then Ø **p**$_{i}$ Ï G, and if Ø **p**$_{i}$ Î G then **p**$_{i}$ Ï G.
    - if y **U** g Î G, then (y Î G or g Î G).
    - if y **R** g Î G, then g Î G.

LTL to BA translation

Now, ***labeling*** of the states of $\mathbf{A}_j$ is defined as:

- The labeling **L: Q**↦S is **L(**G**)={ l | l** Î G Ç S **}**.
  - We want a word s = s$_0$ s$_1$ **…** Î **(**S$_{AP}$**)**^{w} to be in **ℒ(A**$_j$**)** ***iff*** there is a run p = G$_0$ ® G$_1$ ® G$_2$ ® **…** of **A**$_j$ s.t. "**i** Î **ℕ**, we have that s$_i$ "***satisfies***" **L(**G$_i$**)**, i.e., s$_i$ is a "***satisfying assignment***" for **L(**G$_i$**)** .
  - This constitutes a ***slight redefinition of Büchi automata***, where ***labeling is on the states*** instead of on the edges. This facilitates a much more compact **A**$_j$.

LTL to BA translation

Then the ***transition relation*** of $\mathbf{A}_j$.

It is based on the following ***LTL rules***:

- $(\psi\ \mathbf{U}\ \gamma) \equiv \gamma \vee (\psi \wedge \mathbf{X}\,(\psi\ \mathbf{U}\ \gamma))$
- $(\psi\ \mathbf{R}\ \gamma) \equiv \gamma \wedge (\psi \vee \mathbf{X}\,(\psi\ \mathbf{R}\ \gamma)) \equiv (\gamma \wedge \psi) \vee (\gamma \wedge \mathbf{X}(\psi\ \mathbf{R}\ \gamma))$

and on the ***semantics*** of the operator **X**.

- $\mathbf{R} \subseteq \mathbf{Q} \times \mathbf{Q}$, where $(G,G') \in \mathbf{R}$ iff:

  - if $(\psi\ \mathbf{U}\ \gamma) \in G$ then $\gamma \in G$, or $(\psi \in G$ and $(\psi\ \mathbf{U}\ \gamma) \in G')$.
  - if $(\psi\ \mathbf{R}\ \gamma) \in G$ then $\gamma \in G$, and $(\psi \in G$ or $(\psi\ \mathbf{R}\ \gamma) \in G')$.
  - if $\mathbf{X}\ \psi \in G$, then $\psi \in G'$.

LTL to BA translation

- The ***initial states*** of **A** j are **Init = {**G Î **Q** | j Î G**}**.
- The ***accepting states*** of **A** j are defined as follows:

  for each **(**y **U**g) Î **sub(**j **)**, there is a set **F***i* Î **F**, such that:

  - **F***i* = **{**G Î **Q** | **(**y **U** g**)** Ï G or g Î G**}**

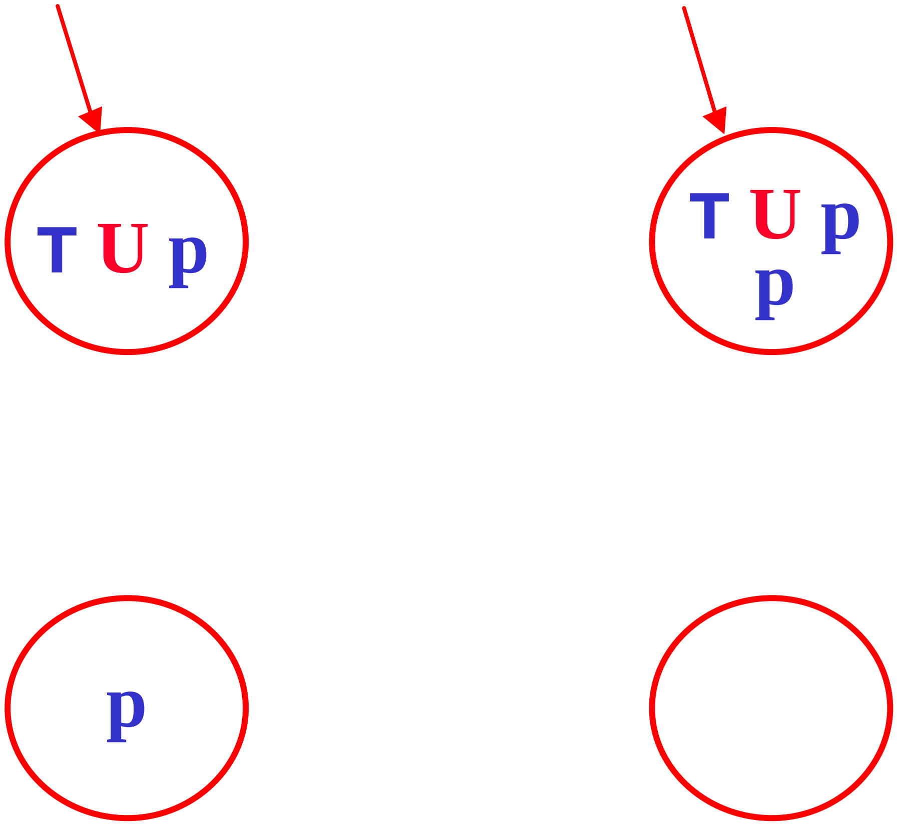
    or equivalently **F***i* = **{**G Î **Q** | if **(**y **U** g**)** Î G, then g Î G**}**

  - Notice that if there is ***no*** **(**y **U** g) Î **sub(**j **)**, then the acceptance condition is the ***trivial acceptance condition***: i.e., ***all states are accepting***

<u>Lemma</u>: $\mathcal{L}$**(** j **)** = $\mathcal{L}$**(A** j **)** .

But **A** j is now a ***<u>generalized</u> Büchi automaton*** …

LTL to BA translation: example

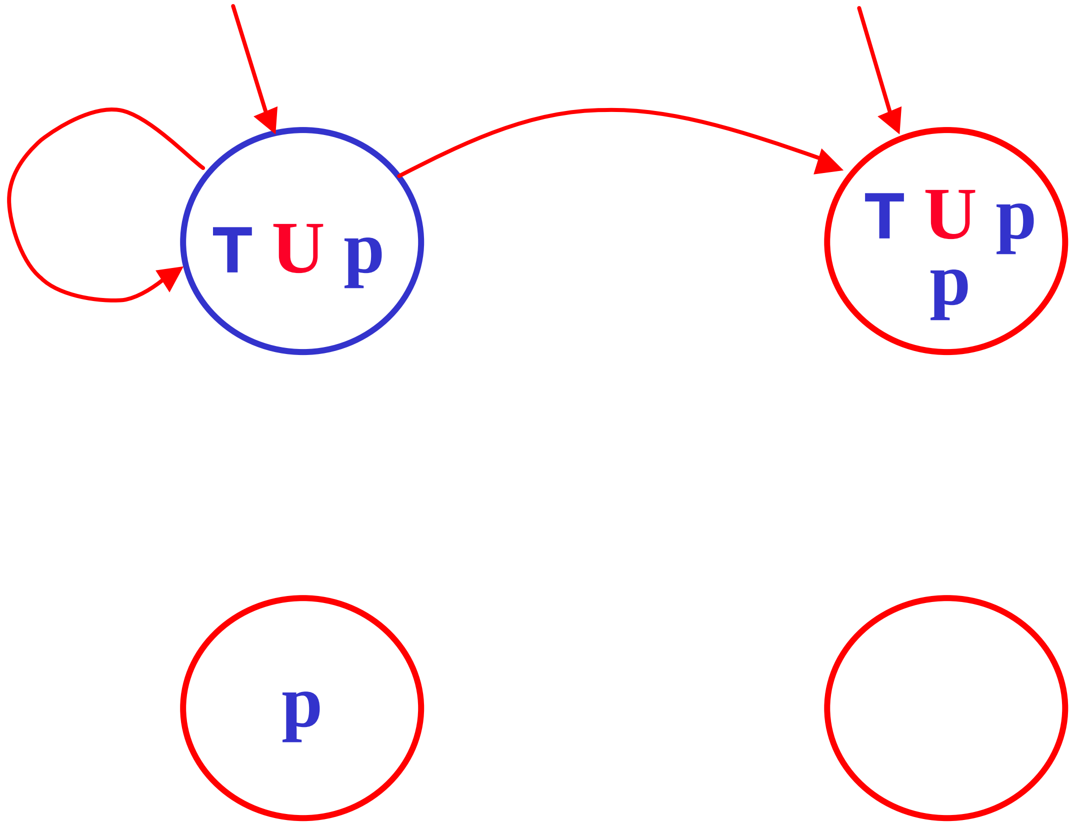


Consider the following formula: $\mathbf{F}\, p \equiv \top\ \mathbf{U}\ p$

$$\mathbf{sub}(\top\ \mathbf{U}\ p) = \{\top\ \mathbf{U}\ p,\ p\}$$

$$\mathbf{Init} = \{\Gamma \in \mathbf{sub}(\top\ \mathbf{U}\ p) \mid \top\ \mathbf{U}\ p \in \Gamma\}$$

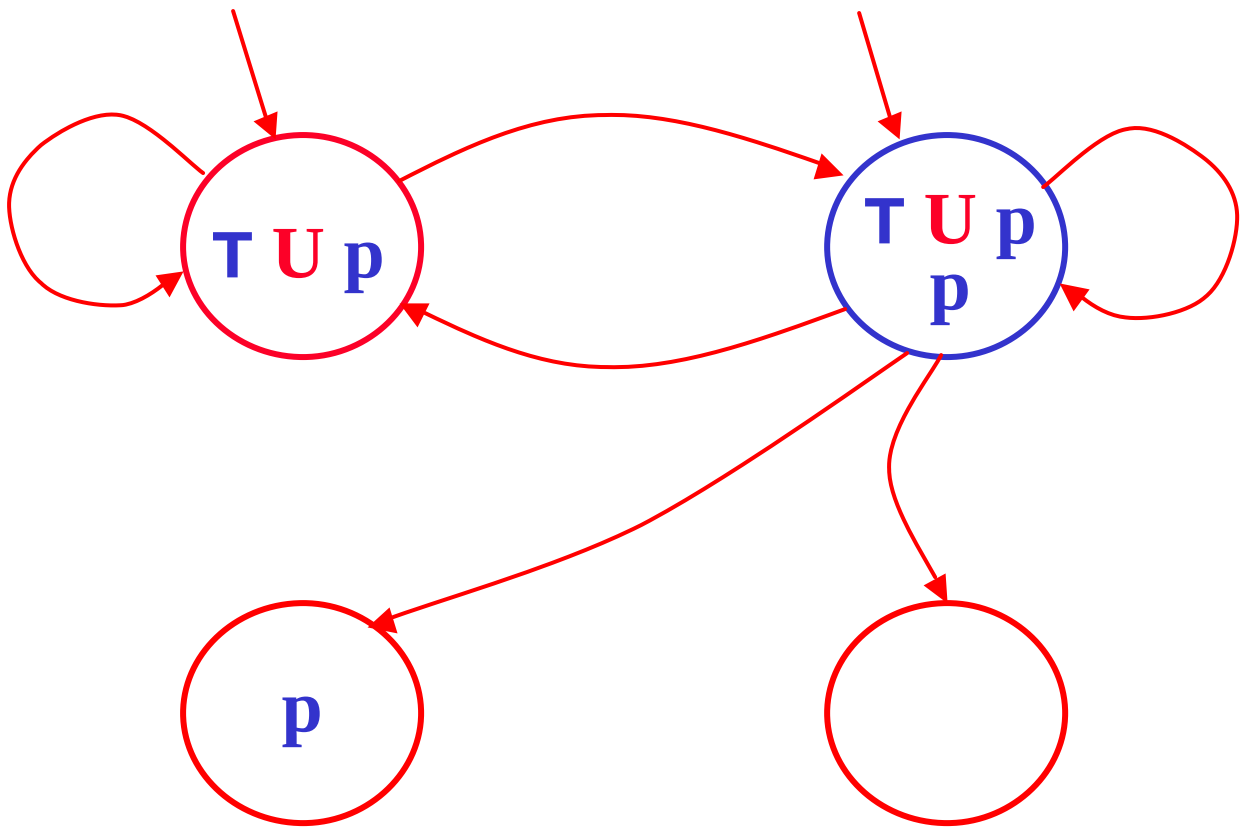
LTL to BA translation: example



Consider the following formula: $\top \, U \, p$

$$(\top \, U \, p) \equiv p \vee X \, (\top \, U \, p)$$

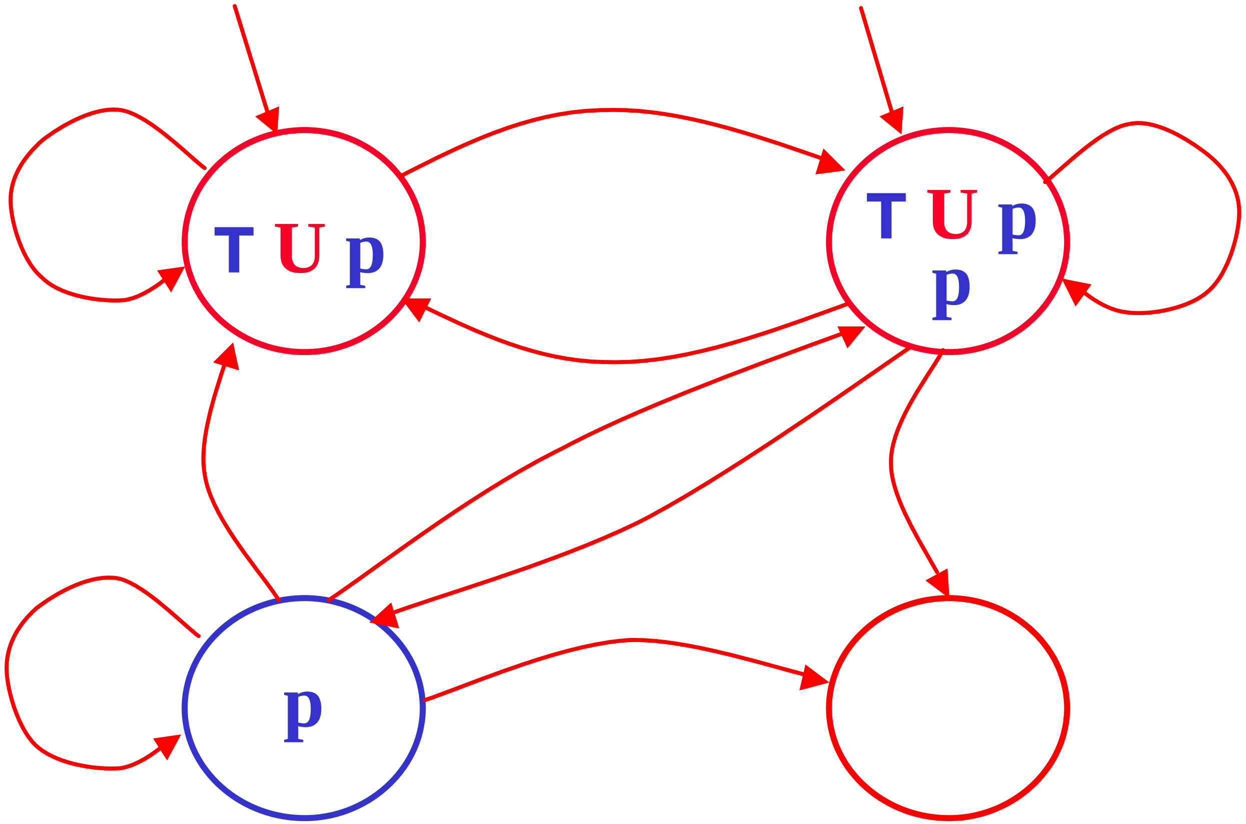
LTL to BA translation: example



Consider the following formula: $\top \, \mathbf{U} \, p$

$$(\top \, \mathbf{U} \, p) \equiv p \lor \mathbf{X} \, (\top \, \mathbf{U} \, p)$$

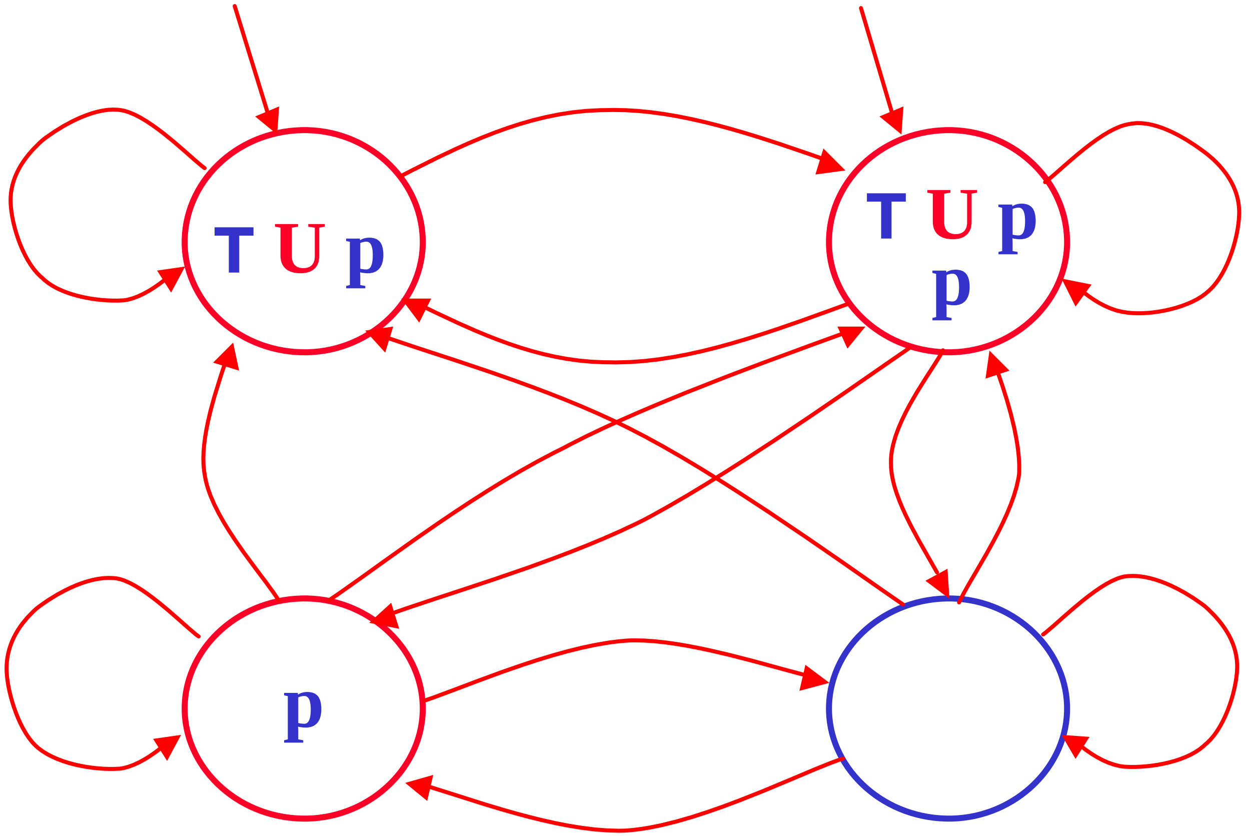
LTL to BA translation: example



Consider the following formula: $\top \; \mathbf{U} \; p$

$$(\top \; \mathbf{U} \; p) \equiv p \lor \mathbf{X} \, (\top \; \mathbf{U} \; p)$$

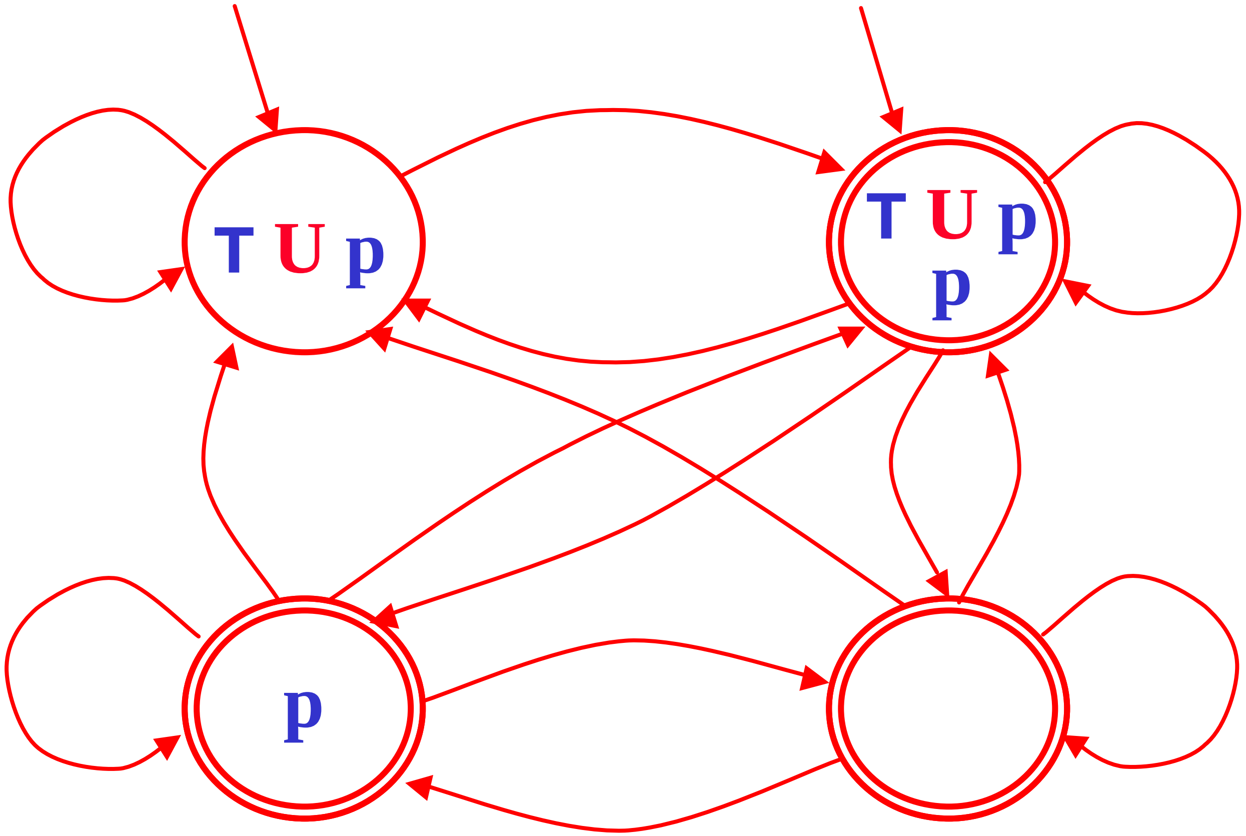
LTL to BA translation: example



Consider the following formula: $\top \, \mathbf{U} \, p$

$$(\top \, \mathbf{U} \, p) \equiv p \vee \mathbf{X} \, (\top \, \mathbf{U} \, p)$$

LTL to BA translation: example

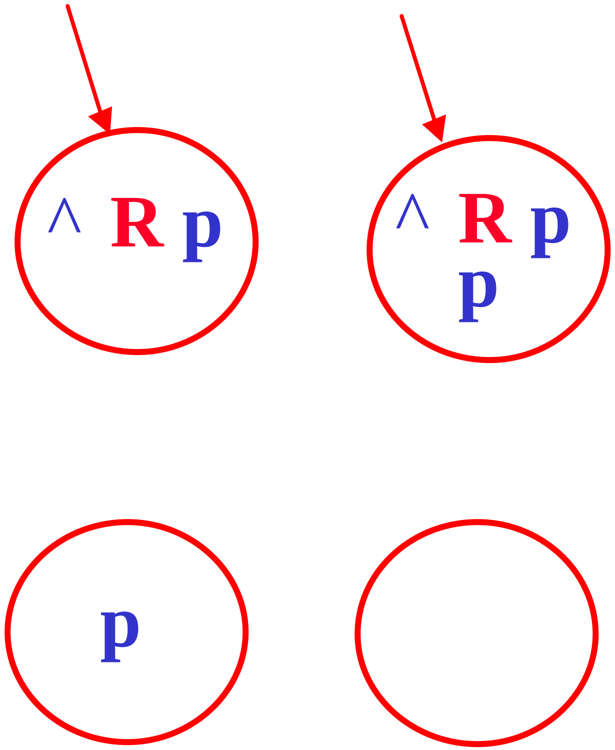


Consider the following formula: $\top\ \mathbf{U}\ p$

$$\mathbf{sub}(\top\ \mathbf{U}\ p) = \{\top\ \mathbf{U}\ p\ ,\ \mathbf{p}\}$$

$$\mathbf{F} = \{\mathbf{F}_{\top \mathbf{U} p}\} = \{G \in \mathbf{sub}(\top\ \mathbf{U}\ p) \mid (\top\ \mathbf{U}\ p) \notin G \text{ or } p \in G\}$$

LTL to BA translation: example

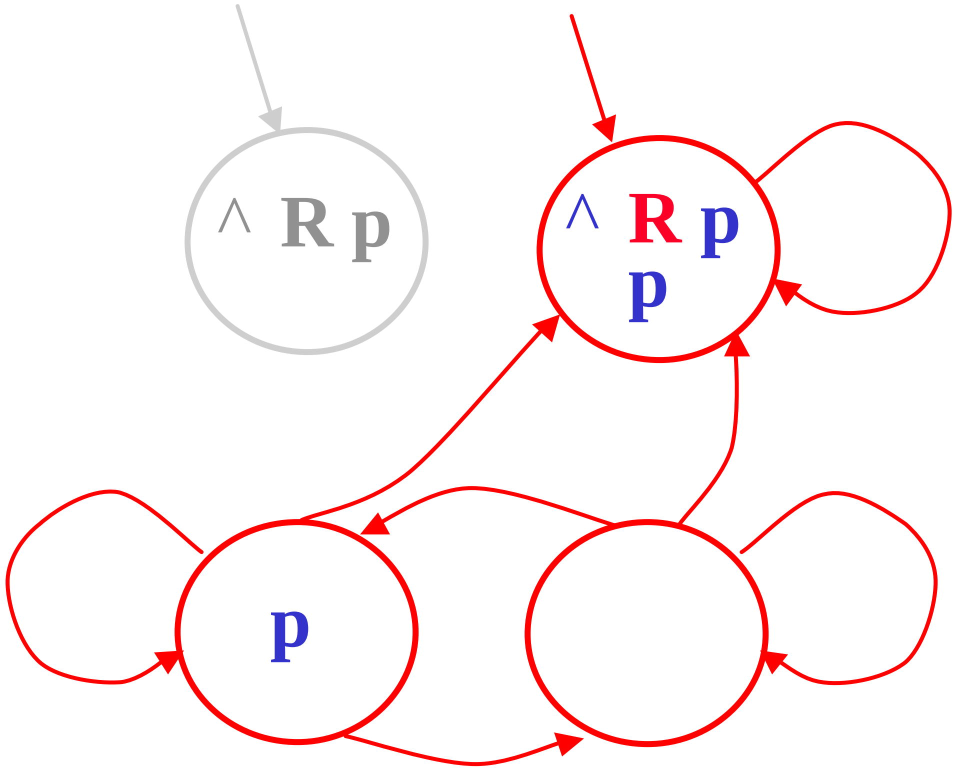


Consider the following formula: G *p* º ^ R *p*

sub(^ R *p*) = {^ R *p* , *p*}

Init = {G Î sub(^ R *p*) | ^ R *p* Î G}

LTL to BA translation: example

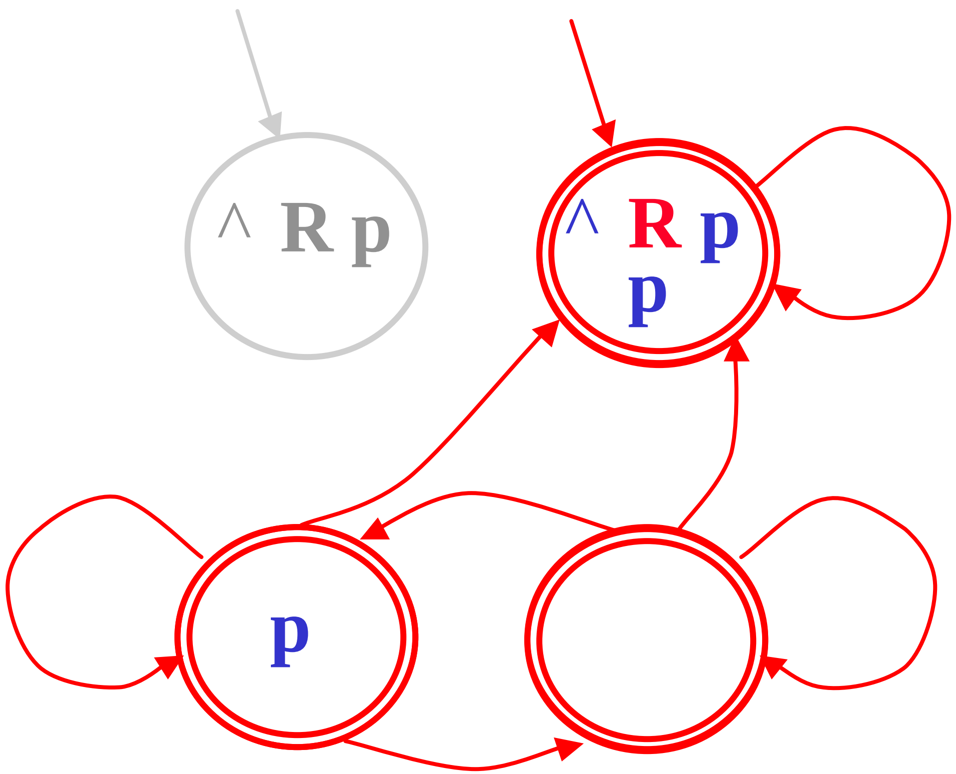


Consider the following formula: G *p* ° ∧ R *p*

sub(∧ R *p*) = {∧ R *p* , *p*}

(∧ R *p*) ° *p* Ù X (∧ R *p*)

LTL to BA translation: example



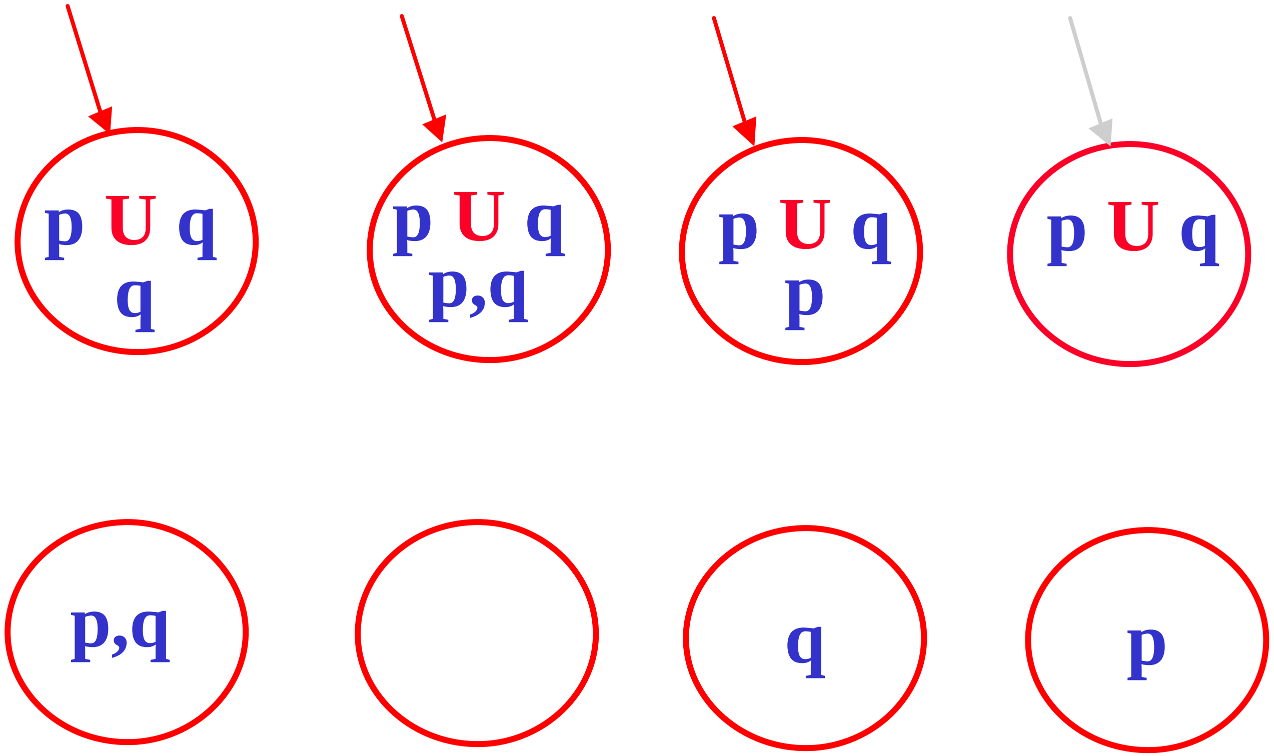
The *trivial* acceptance condition

Consider the following formula: G *p* ° ^ R *p*

sub(^ R *p*) = {^ R *p* , *p*}

There are *no eventualities*, hence F = { Q }

LTL to BA translation: example

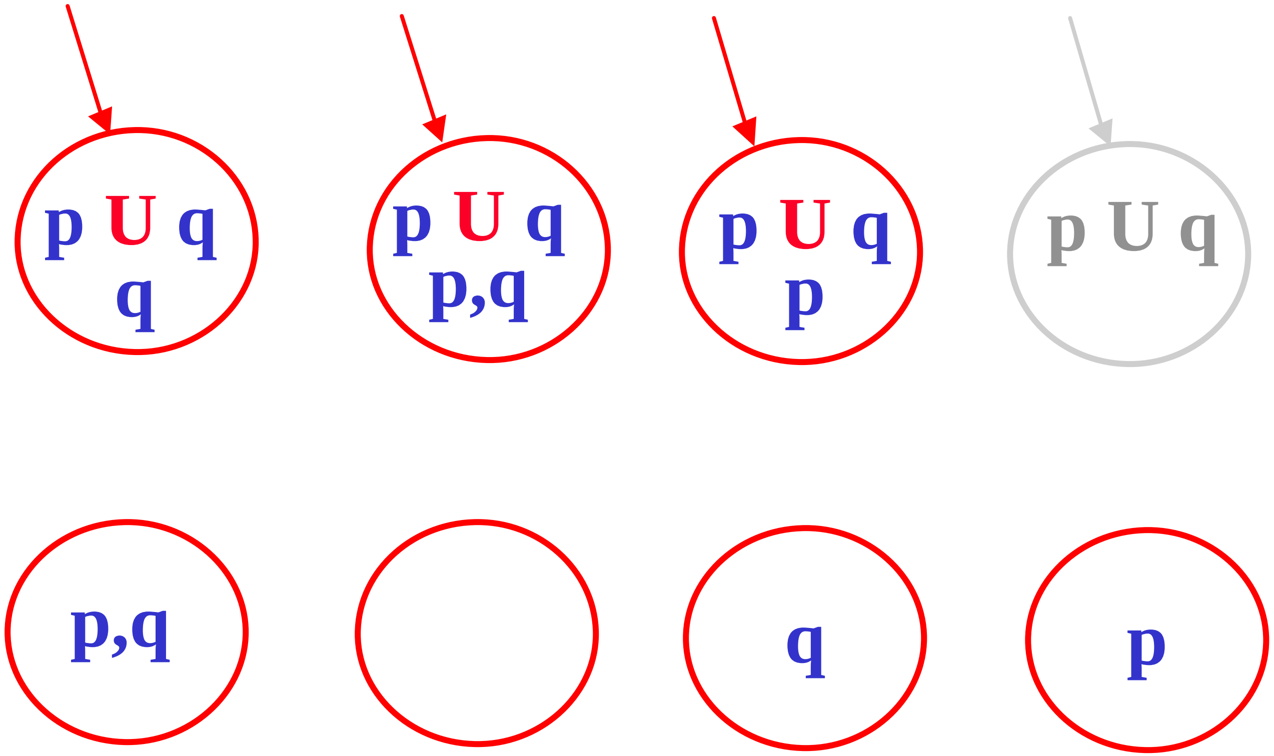


Consider the following formula: ***p* U *q***

sub(*p* U *q*) = {*p* U *q* , *p* , *q*}

Init = {G Î sub(*p* U *p*) | *p* U *q* Î G}

LTL to BA translation: example

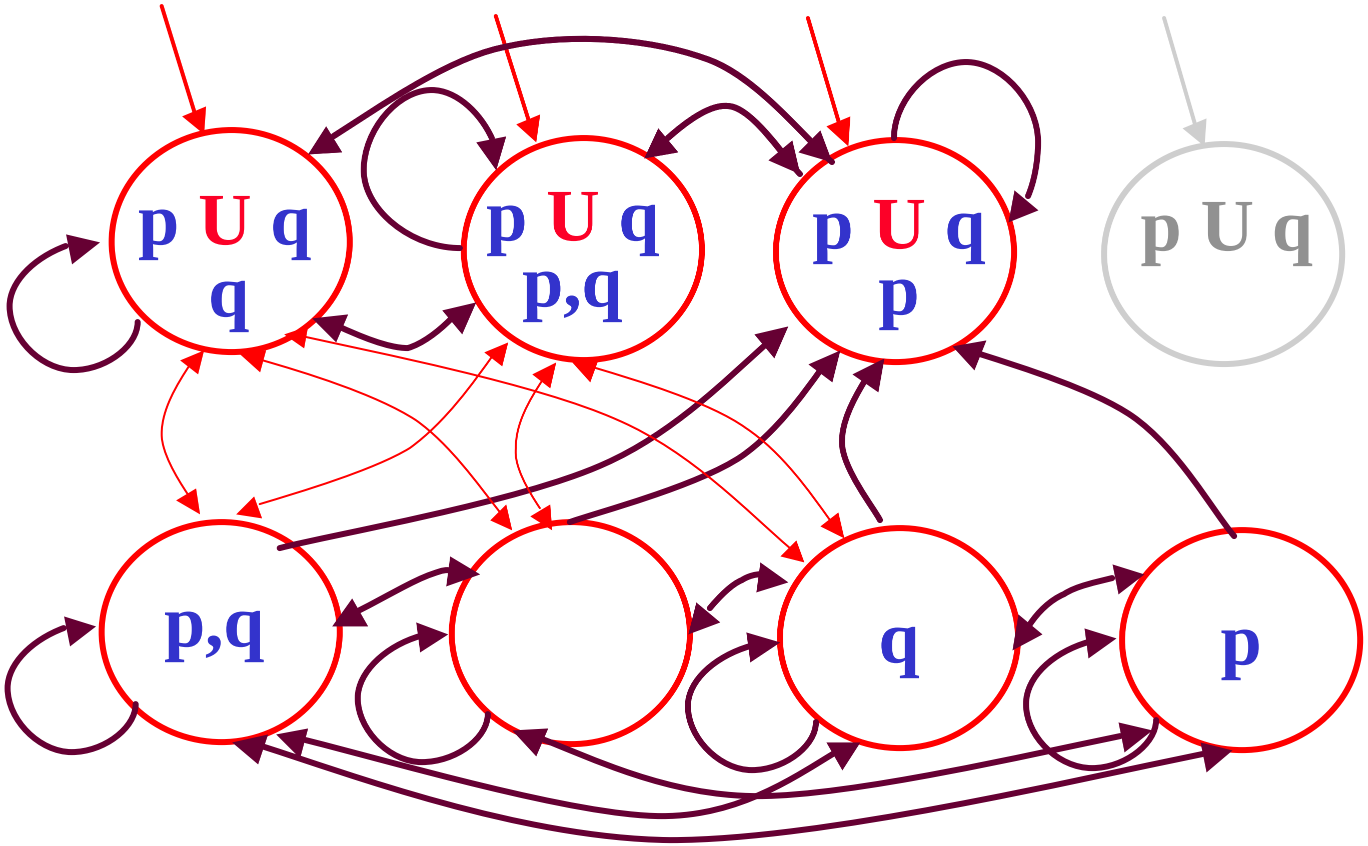


Consider the following formula: $p \mathbin{\mathrm{U}} q$

$$\mathrm{sub}(p \mathbin{\mathrm{U}} q) = \{p \mathbin{\mathrm{U}} q\ ,\ p\ ,\ q\}$$

$$\mathrm{Init} = \{G \in \mathrm{sub}(p \mathbin{\mathrm{U}} p) \mid p \mathbin{\mathrm{U}} q \in G\}$$

LTL to BA translation: example

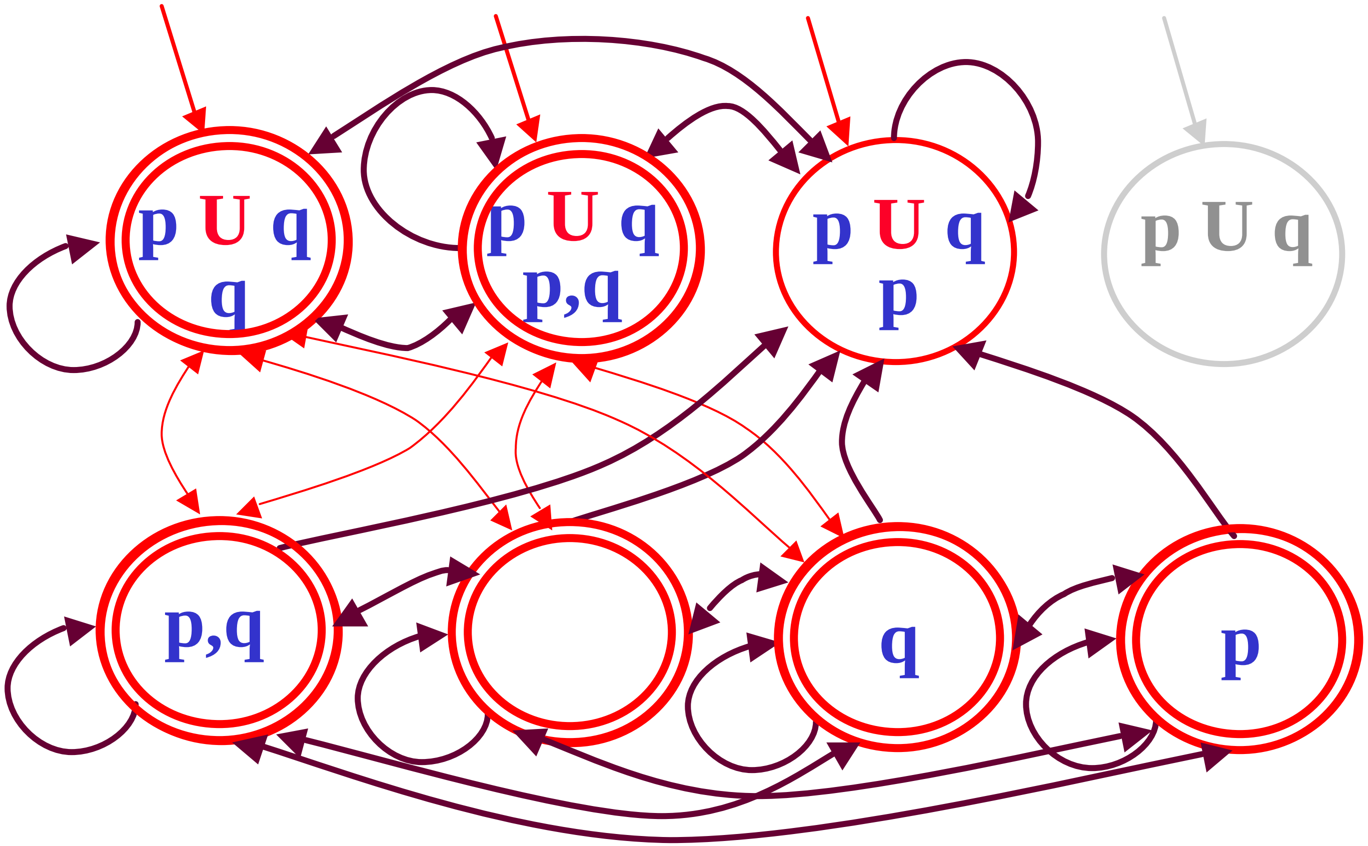


Consider the following formula: ***p* U *q***

sub(*p* U *q*) = {*p* U *q* , *p* , *q*}

(*p* U *q*) ° *q* Ú (*p* Ù X (*p* U *q*))

LTL to BA translation: example



Consider the following formula: ***p*** **U** ***q***

sub(*p* U *q*) = {*p* U *q* , *p* , *q*}

F = { F $_{p\,\mathrm{U}\,q}$ **} = {**G Î **sub(*p* U *q*) | (*p* U *q*)** Ï G or ***q*** Î G}