Model Checking

what is it? And what is it good for?

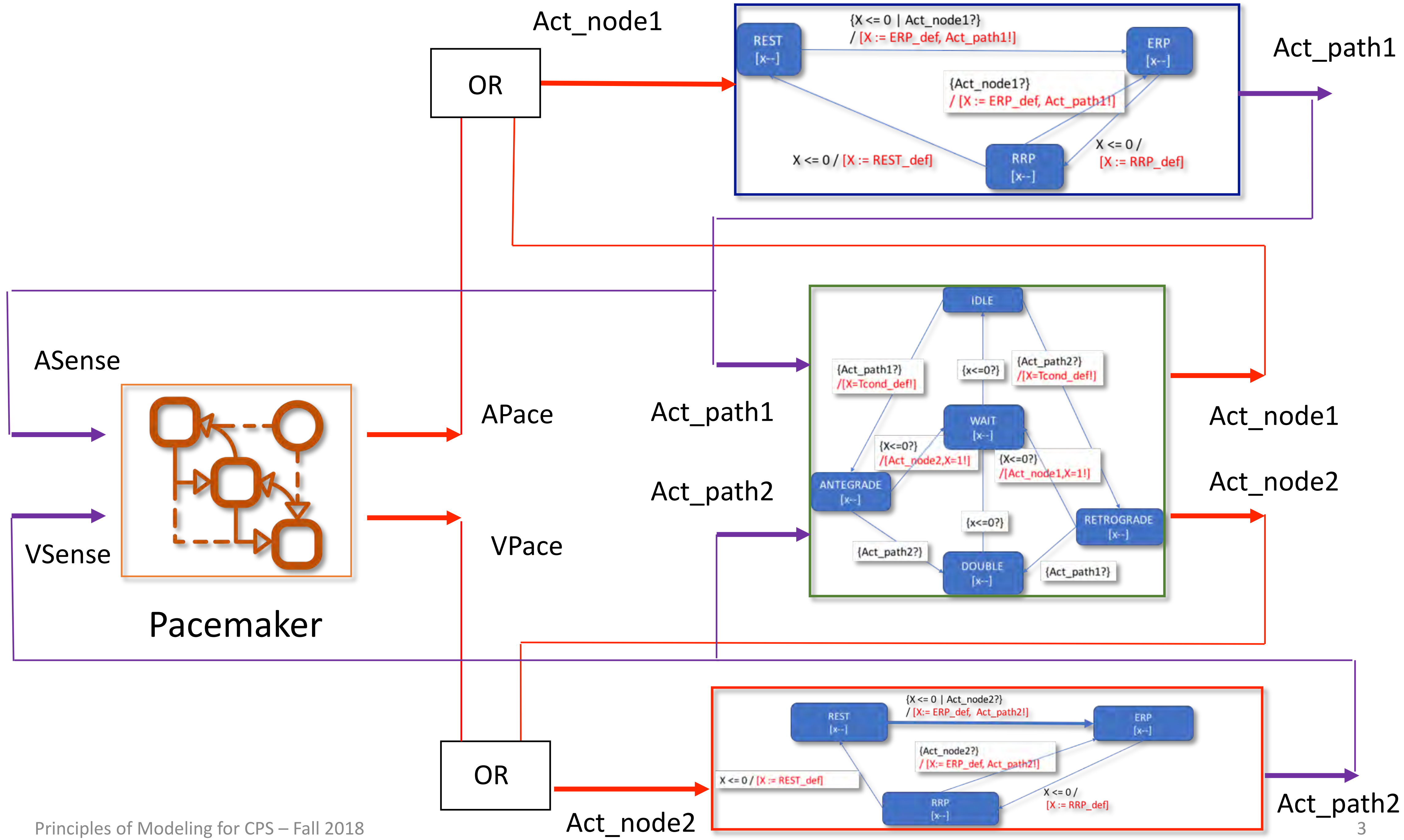
Lecture 14

Principles of Modeling for Cyber-Physical Systems

Instructor: Madhur Behl

So far..

- We modeled the heart (and pacemaker) as a timed automaton with clocks, resets and actions (messages) = timed automaton



Act_node1
OR
{X <= 0 | Act_node1?}
/ [X := ERP_def, Act_path1!]
REST
[x--]
ERP
[x--]
{Act_node1?}
/ [X := ERP_def, Act_path1!]
X <= 0 / [X := REST_def]
RRP
[x--]
X <= 0 /
[X := RRP_def]
Act_path1
ASense
APace
Act_path1
IDLE
{Act_path1?}
/[X=Tcond_def!]
{x<=0?}
{Act_path2?}
/[X=Tcond_def!]
WAIT
[x--]
{X<=0?}
/[Act_node2,X=1!]
{X<=0?}
/[Act_node1,X=1!]
ANTEGRADE
[x--]
RETROGRADE
[x--]
{x<=0?}
{Act_path2?}
DOUBLE
[x--]
{Act_path1?}
Act_node1
Act_path2
Act_node2
VSense
VPace
Pacemaker
OR
Act_node2
{X <= 0 | Act_node2?}
/ [X:= ERP_def, Act_path2!]
REST
[x--]
ERP
[x--]
{Act_node2?}
/ [X:= ERP_def, Act_path2!]
X <= 0 / [X := REST_def]
RRP
[x--]
X <= 0 /
[X := RRP_def]
Act_path2

So far..

- We modeled the heart (and pacemaker) as a timed automaton with clocks, resets and actions (messages) = timed automaton
- The modeling effort allows us to better understand the heart, ask the right questions, and focus on the important aspects for the task at hand.
- Importantly, it allows us to ***automatically and exhaustively check*** whether the heart+pacemaker satisfies some desirable properties.

Automatically and exhaustively

- Importantly, it allows us to *automatically and exhaustively check* whether the heart+pacemaker satisfies some desirable properties.

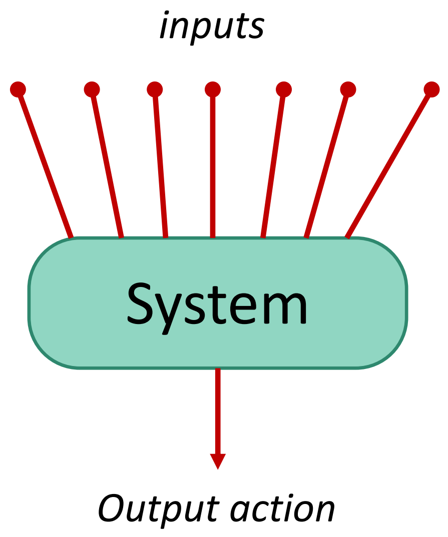
Automatically and exhaustively

- Importantly, it allows us to *automatically and exhaustively check* whether the heart+pacemaker satisfies some desirable properties.
- Automatically: through a computer program
  - You provide a proof of a mathematical theorem…
  - ...vs. the computer provides the proof
- Exhaustively:
  - Testing: simulate the system N times. If testing returns “No bug found”, there could still be a bug (e.g., revealed if you do another N simulations)
  - Exhaustive verification: if the model checker returns “Model is correct”, then this answer is definitive – there is indeed no specification violation. All executions of the model have been *exhaustively* checked.

Next few lectures..

- We explore the basic ideas behind *model checking:* an automatic and exhaustive way of checking whether a system model satisfies some desirable property.
- Our timed automata are more complex than the models we study in this lecture - but what we study forms the basis for understanding all model checking algorithms out there.

Does



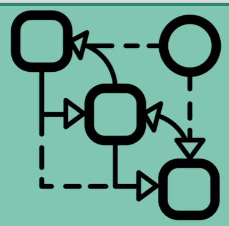
satisfy

Requirements

?

Model Checking UPPAAL Model Checker

Does



Mathematical model

satisfy

$F\ p\ \wedge q$

Formal notation

?

Transition Systems

Linear Temporal Logic

Model Checking

- See ltlmc.ppt

(LTL) Model Checking

Flavio Lerda
with edits by Madhur Behl

Model checking: ingredients

- A *mathematical model* of the system to be verified
- A specification of correct behavior
- Seek to answer: does *every* infinite behavior of the system satisfy the specification?

Ingredients: Heart + pacemaker

- A *mathematical model* of the system: timed automata model of composition of heart + pacemaker
- A specification of correct behavior: e.g., Always, an Asense is followed by another Asense in at most 500ms
- Seek to answer: does *every* infinite behavior of the system satisfy the specification?

Model checking: the question

- Can we answer the question *definitively?* I.e. if the answer is Yes, this is a guarantee that the system model will never produce incorrect behavior.
- Contrast with testing

This lecture

- LTL model checking:
  - The model is a transition system
  - The correct behavior is an LTL formula
- Objective: understand fundamental concepts and uses of model checking

Atomic propositions

- A system model has variables, e.g., voltage.
- An *atomic proposition p* is a statement about the state variable, e.g. p := “voltage > 5” or “-4 <= voltage <= 4”.
- In what follows, AP will denote a set of atomic propositions.

System model: a transition system

- A Transition System (TS) is a tuple $\langle S, I, A, \delta, AP, L\rangle$
  - S is a finite set of states
  - $I \subseteq S$ is a set of initial states
  - A is a finite set of inputs (or `actions')
  - $\delta \subseteq S \times A \times S$ is a transition relation: $s \rightarrow_a s'$
  - AP is a set of atomic propositions on S
  - $L: S \rightarrow 2^{AP}$ is a *state labeling function.* Intuitively, L(s) is the set of atomic propositions satisfied by state s.

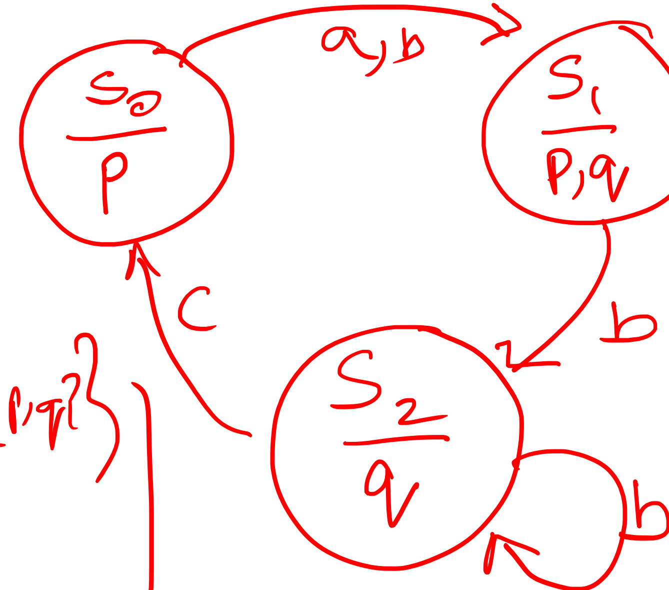
$T = \langle S, I, A, \delta, AP, L \rangle$

$S: \{s_0, s_1, s_2\}$

$I: \{s_0\}$

$A: \{a, b, c\}$

$\delta: \{\langle s_0, a, s_1\rangle, \langle s_1, b, s_2\rangle, \langle s_2, c, s_0\rangle, \langle s_2, b, s_2\rangle\}$



$2^{AP} = \{\{\}, p, q, \{p, q\}\}$

$\Omega = \{x_1 \text{ — } x_k\}$

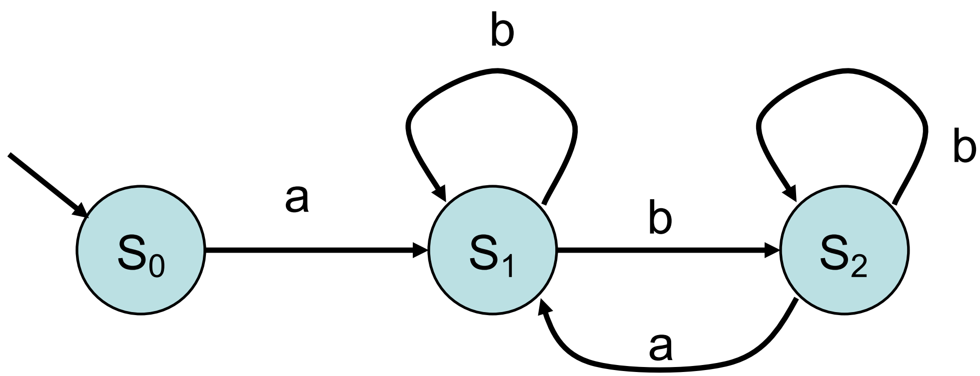
$2^{\Omega} = \{?\} = \{\{\}, \{x_1\} \dots \{x_k\}, \{x_1, x_2\} \dots \{x_{k-1}, x_k\}, \vdots, \{x_1 \dots x_k\}\}$

$AP: \{p, q\}$

$L: S \rightarrow 2^{AP}$

$L(s_0) = p, \quad L(s_1) = \{p, q\}, \quad L(s_2) = q$

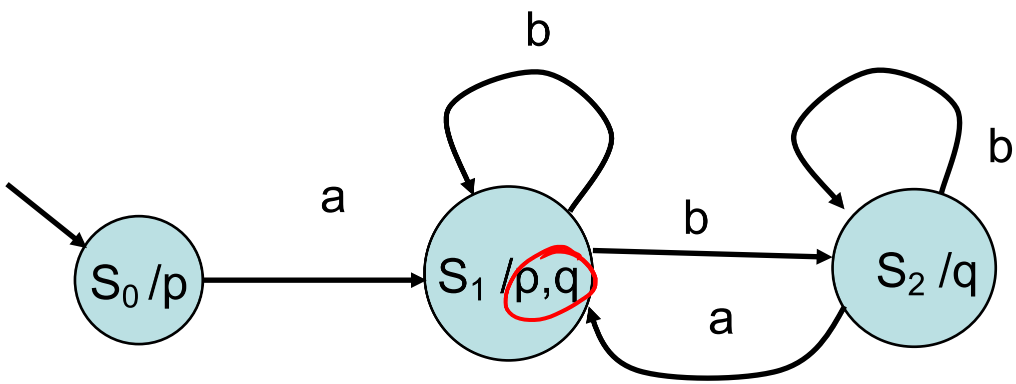
Example transition system



Identify the elements ⟨S, I, A, δ, AP, L⟩ of this transition system

δ {⟨s₀, a, s₁⟩ ⟨s₂, b, s₂⟩
⟨s₁, b, s₁⟩ ⟨s₂, a, s₁⟩}
⟨s₁, b, s₂⟩

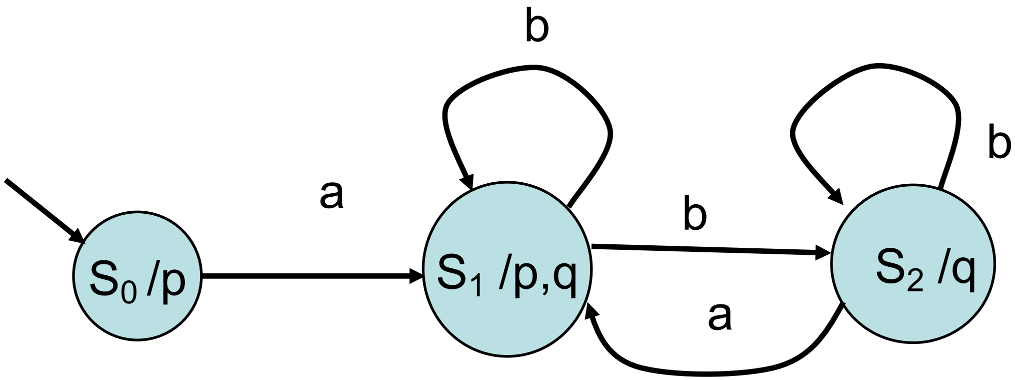
Example transition system



Labeling function: $L(s_0) = p$, $L(s_1) = \{p,q\}$, $L(s_2) = q$

$AP = \{p, q\}$ $L: S \longrightarrow 2^{AP}$

Example transition system

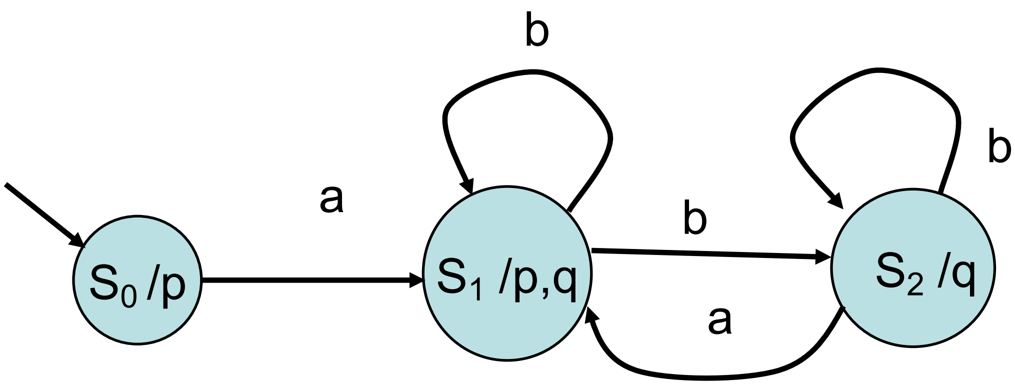


A *path* is an (infinite) sequence of states in the TS.
E.g. $\sigma = S_0S_1S_2S_2S_2S_2\ldots$ is a path in this TS

A *trace* is the corresponding sequence of labels.
E.g. p{p,q}qqqq... Is the trace corresponding to σ

A *word* is a sequence of inputs, e.g. abbbbbb… induces σ

Example transition system



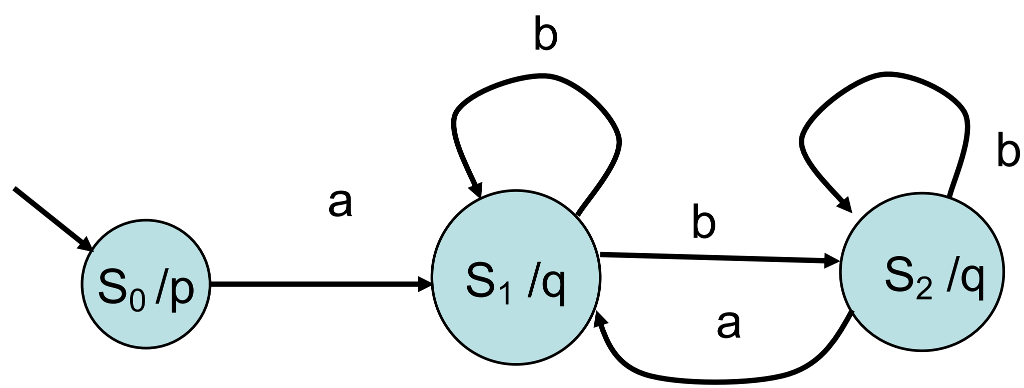
Word abbbbb… gives path $\sigma_1 = S_0 S_1 S_2 S_2 S_2 S_2$…with trace $p\{p,q\}q^+$

Word abbbbb… gives path $\sigma_2 = S_0 S_1 S_1 S_1 S_1 S_1$…with trace $p\{p,q\}^+$

Word ababab… gives path $\sigma_3 = S_0 S_1 S_2 S_1 S_2 S_1$…with trace $p(\{p,q\}q)^*$

Word ababbb… gives path $\sigma_4 = S_0 S_1 S_2 S_1^*$ with trace $p\{p,q\}p\{p,q\}^*$

Example transition system



Word abbbbb… gives path $\sigma_1 = S_0S_1S_2S_2S_2S_2\ldots$with trace pqqq...

Word abbbbb… gives path $\sigma_2 = S_0S_1S_1S_1S_1S_1\ldots$with trace pqqq...

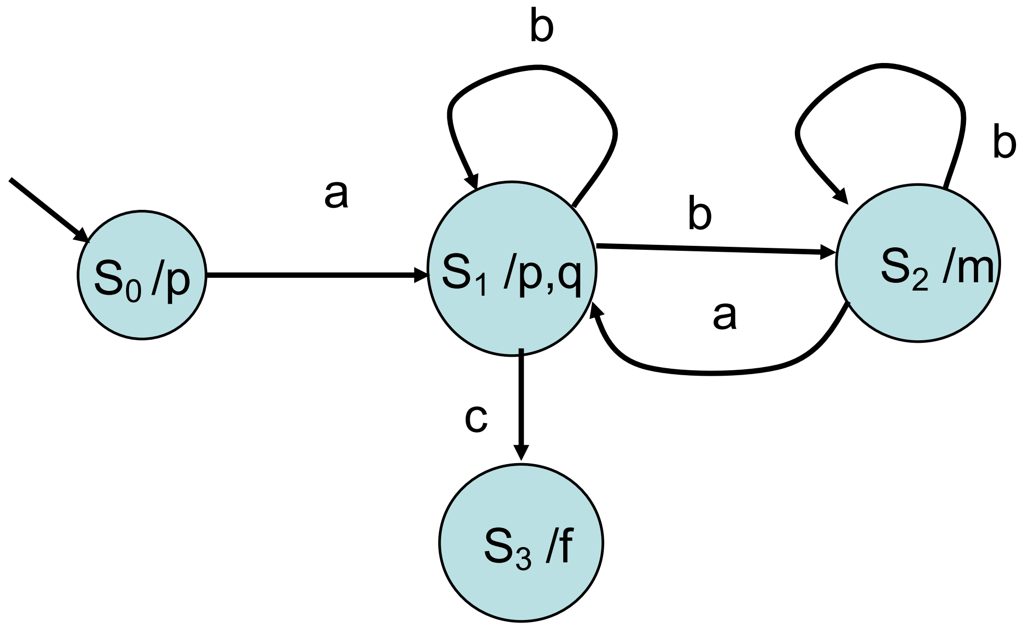
Word ababab… gives path $\sigma_3 = S_0S_1S_2S_1S_2S_1\ldots$with trace pqqq…

Word ababbb… gives path $\sigma_4 = S_0S_1S_2S_1$.. with trace pqqq…

Model checking

- A *mathematical model* of the system to be verified
- A specification of correct behavior
- Seek to answer: does *every* infinite behavior of the system satisfy the specification?

Example specifications



m holds true eventually

m is always followed by q

p holds continuously before f holds

$p((p,q)m)^{+} \rightarrow X$

$p\,\{p,q\}\,m\,\{p,q\}\,f$

Logic

- Rather than focus on specific properties, like those described earlier, and developing custom property-specific checking algorithms…
- Let's define a language for describing *all* (most) properties of interest for systems modeled as transition systems…
- ...then develop an algorithm for checking *any property expressible in this language*.

Linear Temporal Logic (LTL)

- LTL is a logic (a `language') for describing properties of transition systems
- p_k = an *atomic proposition*
- For example, if x is a voltage signal
  - $p_1 := x < 70mV$
  - $p_2 := t > 500ms$
  - $p_3 := \ln(x) > -0.5$
  - $p_4 := e^{ax} + \cos(x) > 45$

b

Linear Temporal Logic (LTL)

- LTL is boolean logic, augmented with two temporal operators: X (next) and U (until)
- An LTL formula is defined inductively as follows:
  - Every atomic proposition p is a formula
  - If φ_1 and φ_2 are LTL formulas, then $\sim\varphi_1$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$ are also LTL formulas
  - $X\ \varphi_1$ is a formula
  - $\varphi_1\ U\ \varphi_2$ is a formula

Boolean Operators

NOT

P	¬
True	
False	

Boolean Operators

NOT

P	¬
True	*False*
False	*True*

Boolean Operators

AND

P	Q	$P \wedge Q$
True	*True*	
True	*False*	
False	*True*	
False	*False*	

Boolean Operators

AND

P	**Q**	**P $\wedge$ Q**
True	*True*	*True*
True	*False*	*False*
False	*True*	*False*
False	*False*	*False*

Boolean Operators

OR

P	**Q**	**P $\vee$ Q**
True	*True*	
True	*False*	
False	*True*	
False	*False*	

Boolean Operators

OR

P	Q	$P \vee Q$
True	*True*	*True*
True	*False*	*True*
False	*True*	*True*
False	*False*	*False*

Boolean Operators

IMPLIES

P	**Q**	**P $\rightarrow$ Q**
True	*True*	
True	*False*	
False	*True*	
False	*False*	

Boolean Operators

IMPLIES

P	Q	P → Q
True	*True*	*True*
True	*False*	*False*
False	*True*	*True*
False	*False*	*True*

So p → q follows the following reasoning:

1. a True premise implies a True conclusion, therefore T → T is T;
2. a True premise cannot imply a False conclusion, therefore T → F is F; and
3. you can conclude anything from a false assumption, so F → anything is T.

Linear Temporal Logic (LTL)

- LTL is boolean logic, augmented with two temporal operators: X (next) and U (until)
- An LTL formula is defined inductively as follows:
  - Every atomic proposition p is a formula
  - If φ_1 and φ_2 are LTL formulas, then $\sim\varphi_1$,
  - $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$ are also LTL formulas
  - $X\ \varphi_1$ is a formula
  - $\varphi_1\ U\ \varphi_2$ is a formula

LTL semantics intuition (slide courtesy of G. Fainekos at ASU)

p – p now

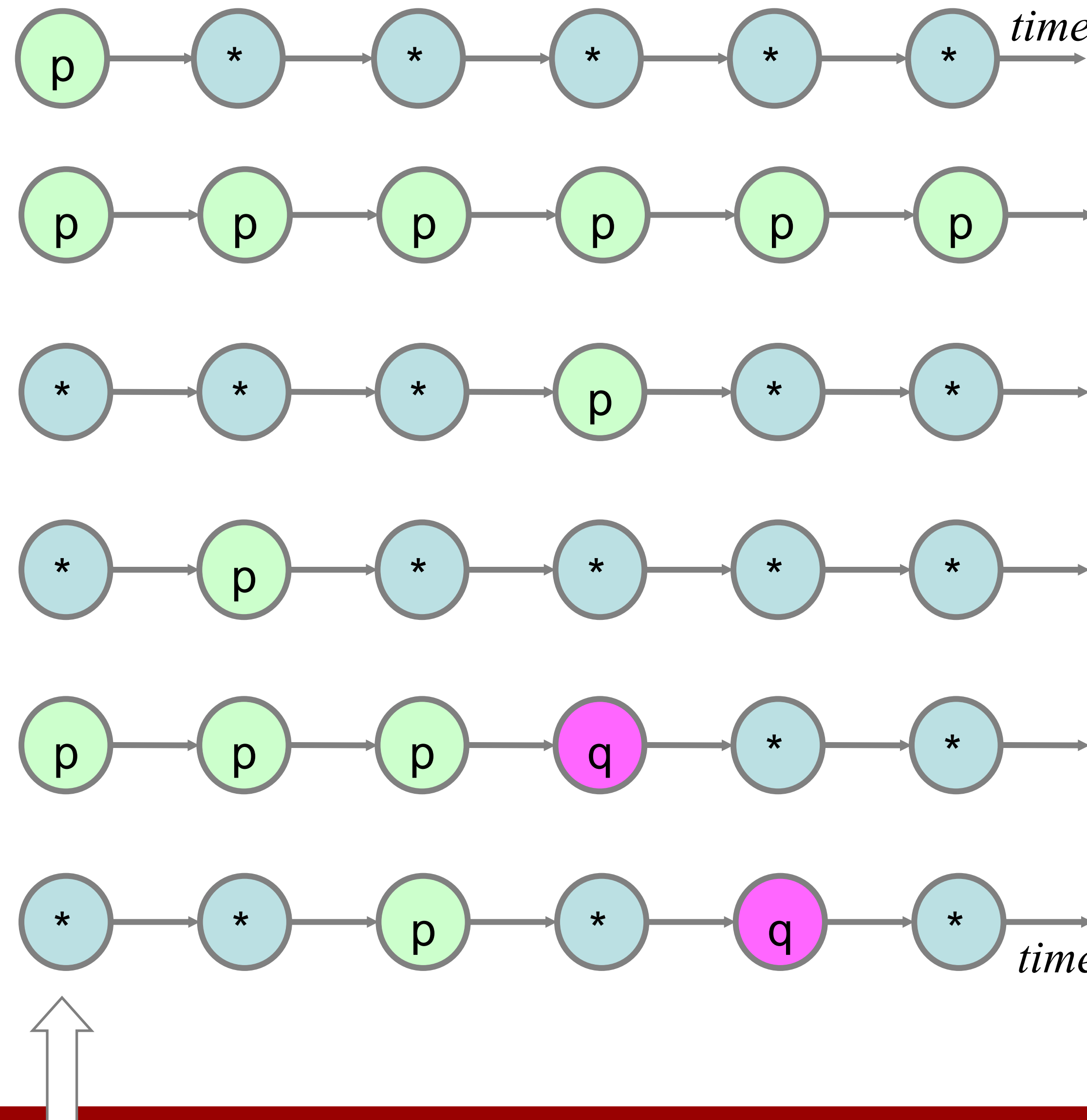
G p - always p

F p – eventually p

X p – next state p

p $\mathcal{U}$ q – p until q

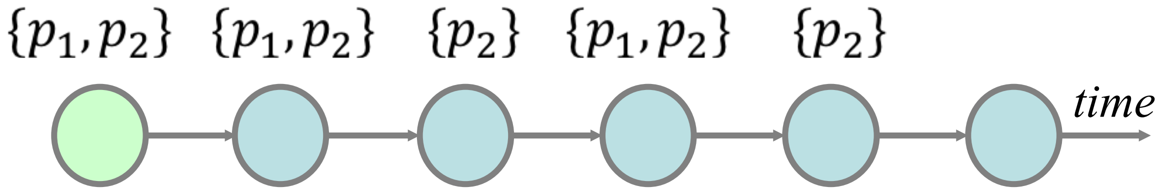
p $\mathcal{B}$ q – p before q



p_2

p_1

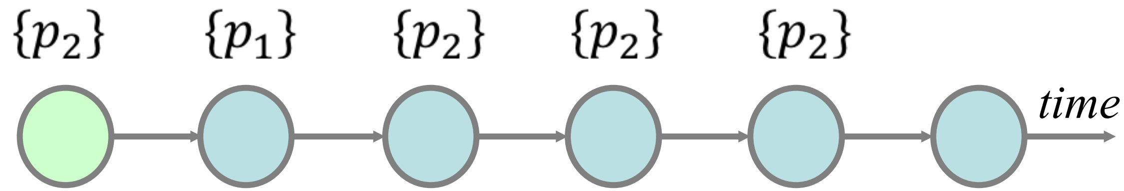
$p_1 \wedge p_2$



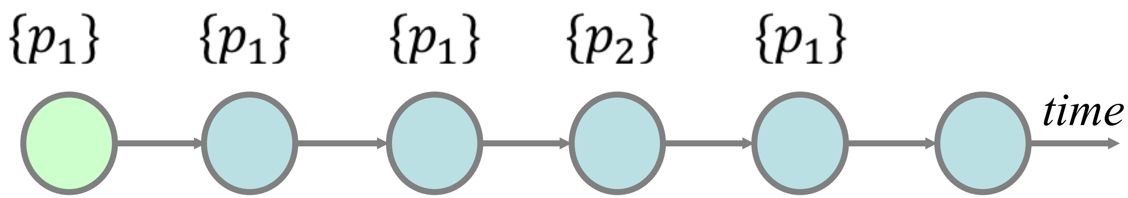
$\neg p_1$

$X p_1$

$X(p_1 \wedge \neg p_2)$



$p_1 \cup p_2$



$$\phi \coloneqq true \mid p_1 \mid \emptyset_1 \wedge \emptyset_2 \mid \neg\emptyset_1 \mid X\emptyset \mid \emptyset_1 \cup \emptyset_2$$

$p_i \in AP$

$\emptyset_1, \emptyset_2 : LTL\ formulas$

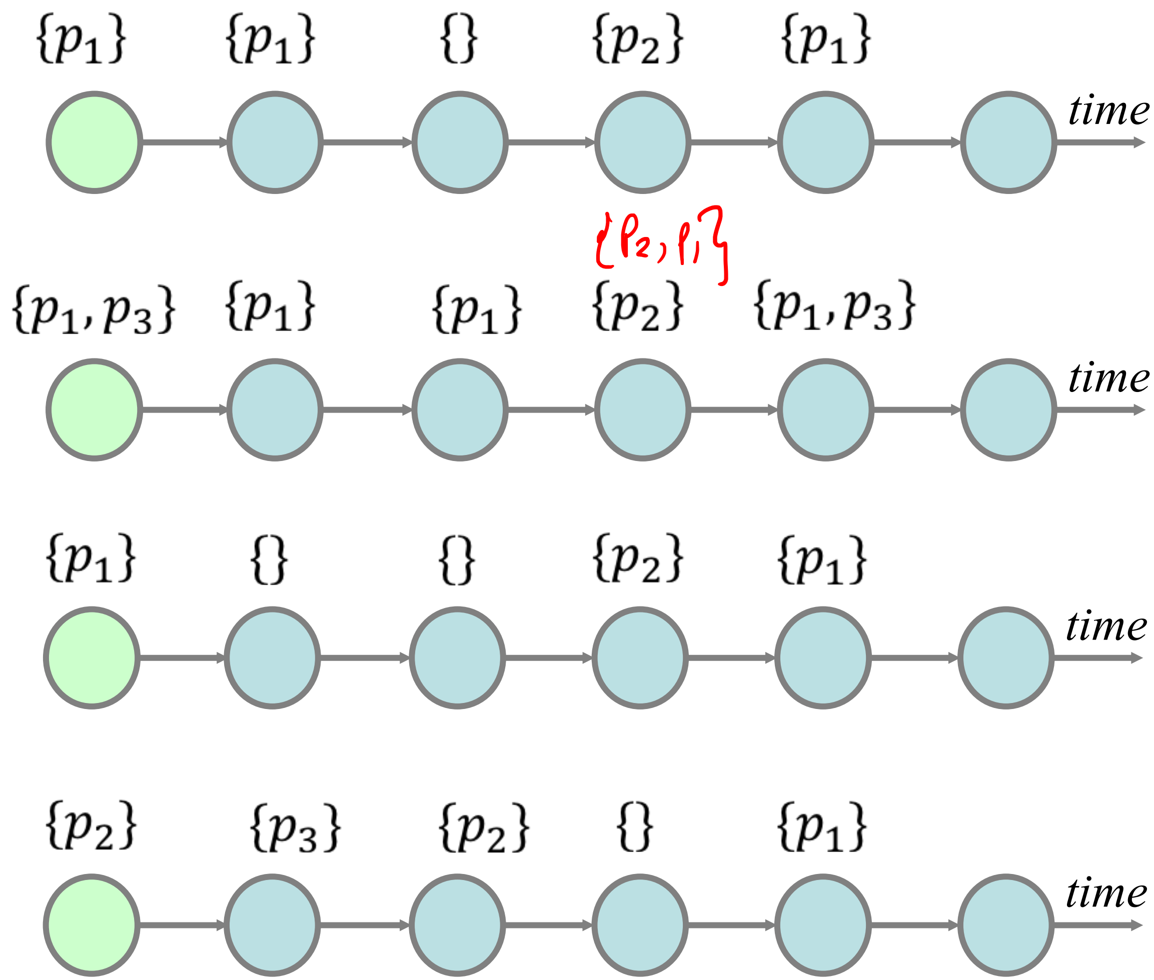
$$\phi := true \mid p_1 \mid \emptyset_1 \land \emptyset_2 \mid \neg\emptyset_1 \mid X\emptyset \mid \emptyset_1 \cup \emptyset_2$$

$\neg(p_1 \cup p_2)$

$p_1 \cup (p_2 \land Xp_3)$

$X(\neg p_1 \cup p_2)$

$true \cup p_1$



F p_1

$G\ p_1$

Globally

$p_1\ U\ \text{false}$ ✓ x

$p_1\ U\ (\sim\text{true})$

$\sim(F\ \sim p_1)$ x

$\sim(\text{true}\ U\ \sim p_1)$

$\sim\text{true}\ U\ p_1) \Rightarrow \text{false}\ U\ p_1$ x

$\sim(\underbrace{\text{true}\ U\ p_1}_{}) \rightarrow \sim Fp \rightarrow qqq^{+}$ x

Fp

$(p_1\ U\ \text{true}) \rightarrow \underbrace{p_1 p_1 p_1 qqq^{+}}$ x

$p_1\ U\ p_1 \rightarrow p$

$p_1\ X\ p_1 \longrightarrow p_1 p_1 q^{+}$

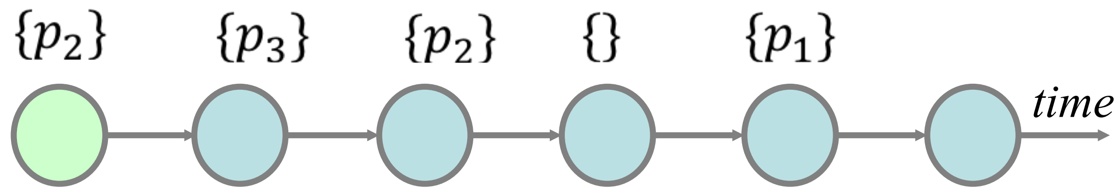
$p_1 \wedge G\cancel{x}$

$\underbrace{p_1\ B\ \text{true}} \rightarrow$

Derived formulae

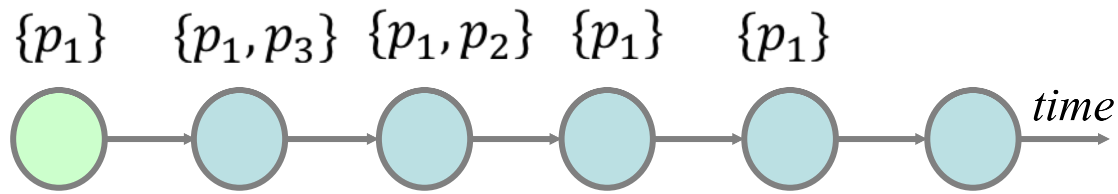
$F\ p_1$ $\quad$ $true \cup p_1$

Eventually



$G\ p_1$ $\quad$ $\neg(true \cup \neg p_1)$

Globally



$\emptyset_1 \vee \emptyset_2$ $\quad$ $\neg(\neg\emptyset_1 \wedge \neg\emptyset_2)$

Or

$\emptyset_1 \rightarrow \emptyset_2$ $\quad$ $\neg\emptyset_1 \vee \emptyset_2$

Implies

Linear Temporal Logic (LTL)

- LTL is boolean logic, augmented with two temporal operators: X (next) and U (until)
- An LTL formula is defined inductively as follows:
  - Every atomic proposition p is a formula
  - If φ_1 and φ_2 are LTL formulas, then $\sim\varphi_1$,
  - $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$ are also LTL formulas
  - $X\ \varphi_1$ is a formula
  - $\varphi_1\ U\ \varphi_2$ is a formula

Notation

- Sometimes you'll see alternative notation in the literature:

G	$\Box$
F	$\Diamond$
X	$\circ$

Examples of LTL formulas

- Invariant (something always holds) :
  - G(~p) (~ is negation)
- Response
  - G (p → F q)
- Fairness
  - (G F p) → (G F q)

Examples of LTL formulas

- Invariant (something always holds) :
  - G(~p) (~ is negation)

Safety:

"something bad will not happen"

$$\Box \neg (reactor\_temp > 1000)$$

Examples of LTL formulas

Liveness:

"something good will happen"

Typical examples:

$\Diamond rich$

$\Diamond(x > 5)$

$\Box(start \Rightarrow \Diamond terminate)$

and so on.....

Usually: $\Diamond$....

Examples of LTL formulas

Often only really useful when scheduling processes, responding to messages, etc.

Strong Fairness:

"if something is attempted/requested infinitely often, then it will be successful/allocated infinitely often"

Typical example:

$$\Box\Diamond ready \Rightarrow \Box\Diamond run$$

Examples of LTL formulas

An LTL formula is defined inductively as follows:

- Every atomic proposition p is a formula
- If φ_1 and φ_2 are LTL formulas, then $\sim\varphi_1$, $\varphi_1 \vee \varphi_2$,
- $\varphi_1 \wedge \varphi_2$ are also LTL formulas
- $X\ \varphi_1$ is a formula
- $\varphi_1\ U\ \varphi_2$ is a formula

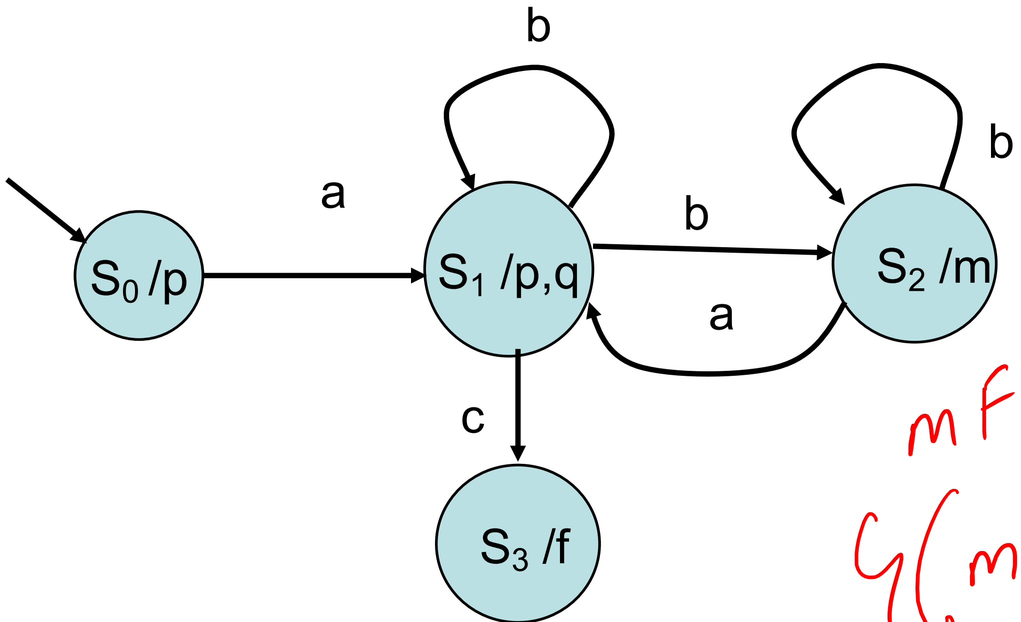
•Which of these are valid LTL formulas?

- $\sim\sim\varphi_1$: $\to \phi \to \sim(\phi_2)$ $\quad \phi_2 = \sim\phi_1$
- $\sim(\varphi_1\ \cancel{U}U\ \varphi_2)$
- $G(\sim\varphi_1 \vee \sim\varphi_1)$

$\hookrightarrow G\phi_2 \to \phi_2 = Q_3 \vee Q_4 \quad Q_3, Q_4 > \sim Q_5, Q_6$

STUDENT

Example specifications in LTL



$m\,F\,X\,F\,q$

$G(m \rightarrow Fq)$

$G(m \rightarrow Xq)$

$m \rightarrow Xq$

Express these in LTL:

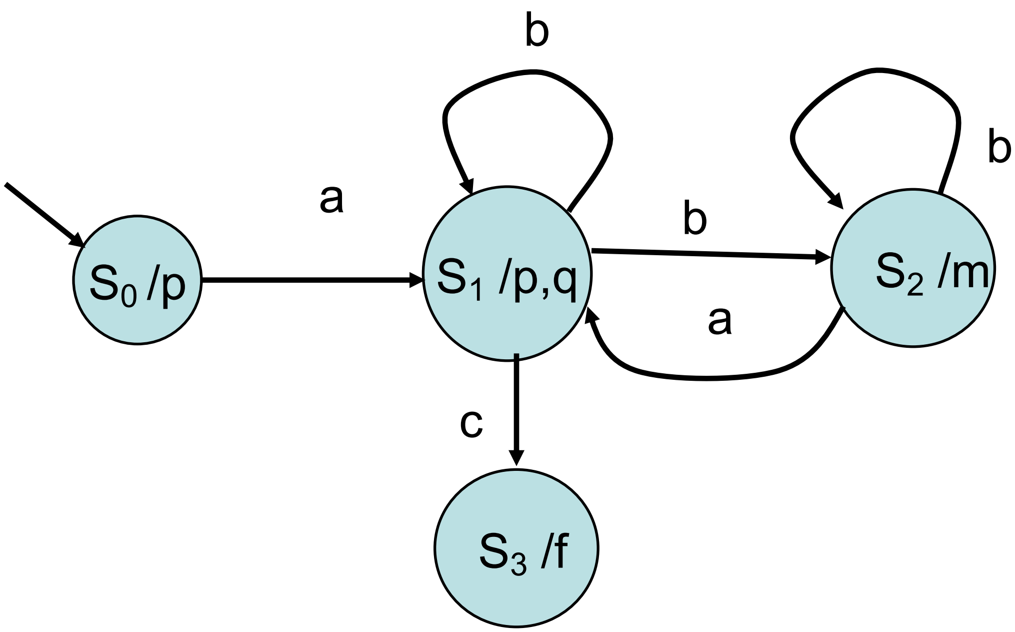
m holds true eventually → Fm

Always, m holds true eventually → GFm

m is always followed by q → $G(m \rightarrow q)$

p holds continuously before f holds pUf

Example specifications in LTL



Express these in LTL:

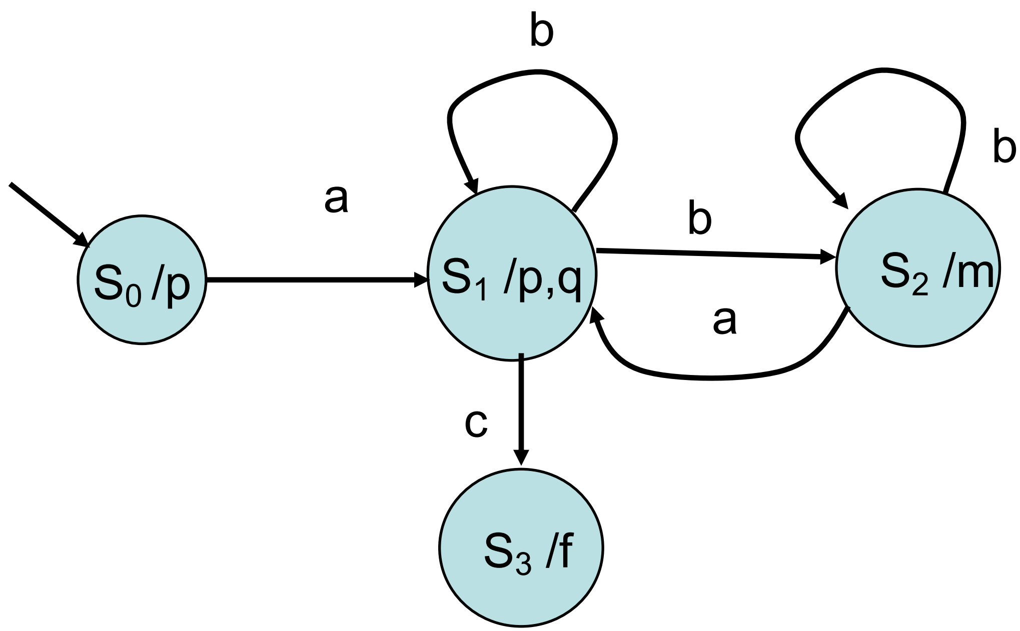
m holds true eventually: Fm

Always, m holds true eventually : GFm

m is always followed by q : G(m → X q)

p holds true continuously before f holds true: p U f

Example specifications in LTL



Does the TS satisfy these specifications:

m holds true eventually: Fm

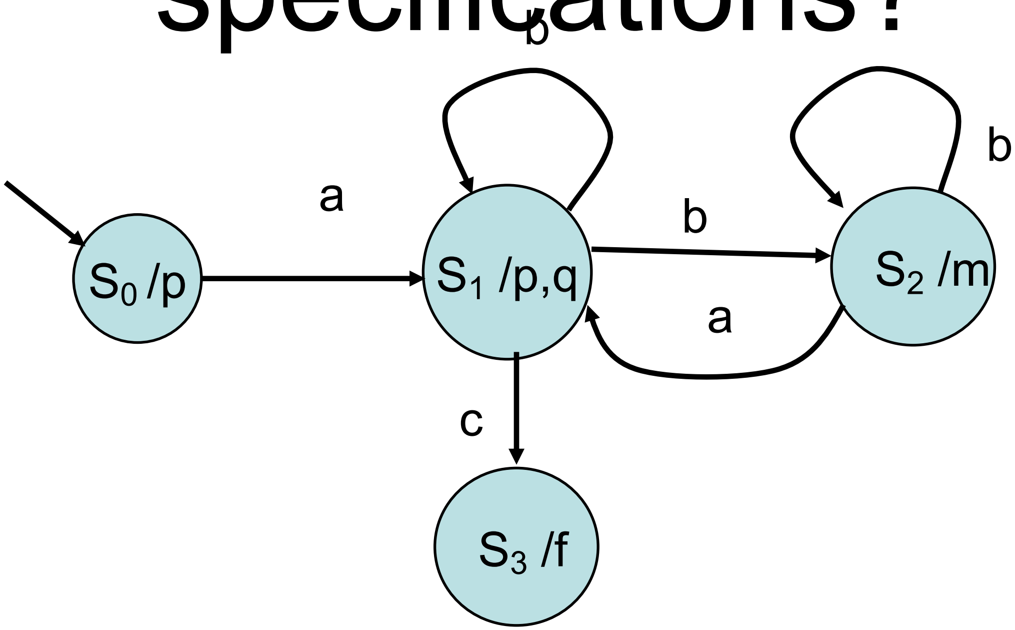
Always, m holds true eventually : GFm

m is always followed by q : G(m → X q)

p holds true continuously before f holds true: p U f

STUDENT

Does the TS satisfy these specifications?



Does the TS satisfy these specifications:

m holds true eventually: Fm : No

Always, m holds true eventually : GFm : No

m is always followed by q : G(m → X q): No

p holds continuously before f holds: p U f: No

Announcements

- No Lectures next week ! (Conference travel)
- Assignment 5 deadline has been extended from Tuesday, Nov 6 to Thursday, Nov 8m 11:59pm.

- A Simulink/Stateflow walkthrough video will be posted in lieu of the lectures next week. It will help with assignment 5.
- Assignment 6 on transition systems and LTL will be out next week on Thursday, Nov 8. It is due in 1 week – on Thursday, Nov 15, at 2:00pm (before the lecture).

LTL to Buchi automata

- We have a system model as a transition system (TS), aka an *automaton.*
- And a specification as an LTL formula
- Recall design principle: try to stick to the same formalism.

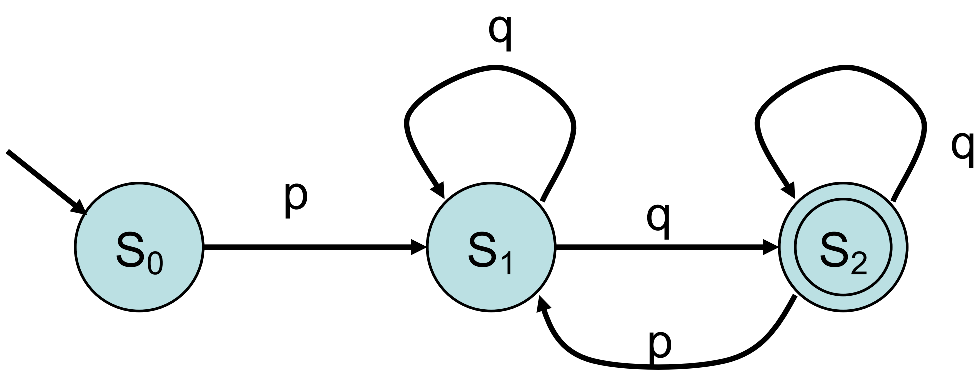
LTL to Buchi automata

- We have a system model as a transition system (TS), aka an *automaton.*
- And a specification as an LTL formula
- Recall design principle: try to stick to the same formalism.
- Every LTL formula has a corresponding *Buchi automaton* that accepts all and only the infinite state traces that satisfy the formula [Vardi and Wolper]

Büchi Automaton

- Automaton which accepts infinite paths
- A Büchi automaton is tuple $\langle S, I, A, \delta, F\rangle$
  - S is a finite set of states (like a TS)
  - $I \subseteq S$ is a set of initial states (like a TS)
  - A is a finite alphabet (like a TS)
  - $\delta \subseteq S \times A \times S$ is a transition relation (like a TS)
  - $F \subseteq S$ is a set of accepting states
- An infinite sequence of states (a *path*) is accepted iff it contains accepting states (from F) infinitely often

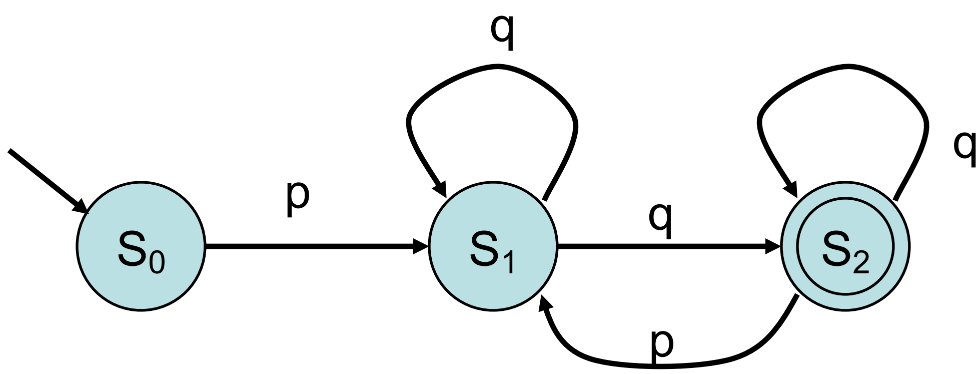
Identify Büchi Automaton components



STUDENT

- A Büchi automaton is tuple $\langle S, I, A, \delta, F\rangle$
  - S is a finite set of states (like a TS)
  - $I \subseteq S$ is a set of initial states (like a TS)
  - A is a finite alphabet (like a TS)
  - $\delta \subseteq S \times A \times S$ is a transition relation (like a TS)
  - $F \subseteq S$ is a set of accepting states

Example: accepted paths

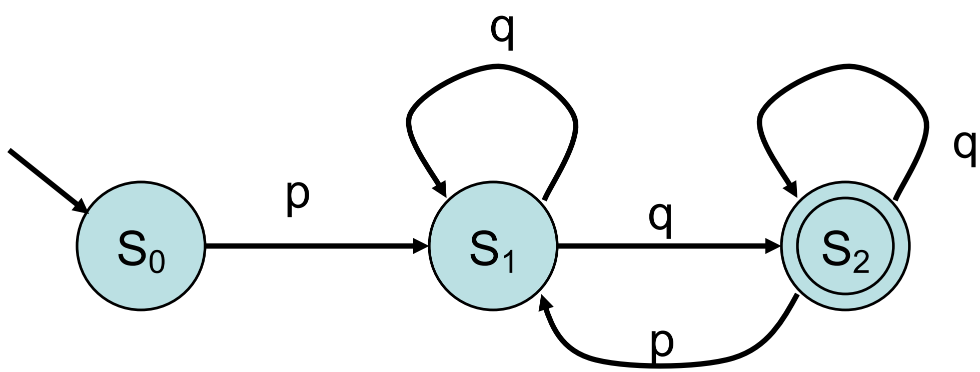


$\sigma_1 = S_0 S_1 S_2 S_2 S_2 S_2 \ldots$ **ACCEPTED**

$\sigma_2 = S_0 S_1 S_2 S_1 S_2 S_1 \ldots$ **ACCEPTED**

$\sigma_3 = S_0 S_1 S_2 S_1 S_1 S_1 \ldots$ **REJECTED**

Example: accepted words



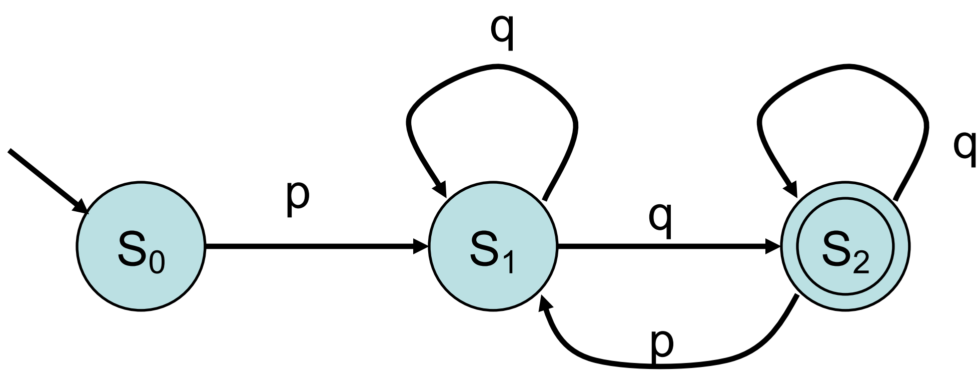
Automaton B = $\langle S, I, A, \delta, F\rangle$

Word = infinite sequence of letters from alphabet A.
E.g. pq^+ and $p(q^*qp)^*$ are both words.

What words are accepted by this automaton?

STUDENT

Example



Word = infinite sequence of letters from alphabet A .

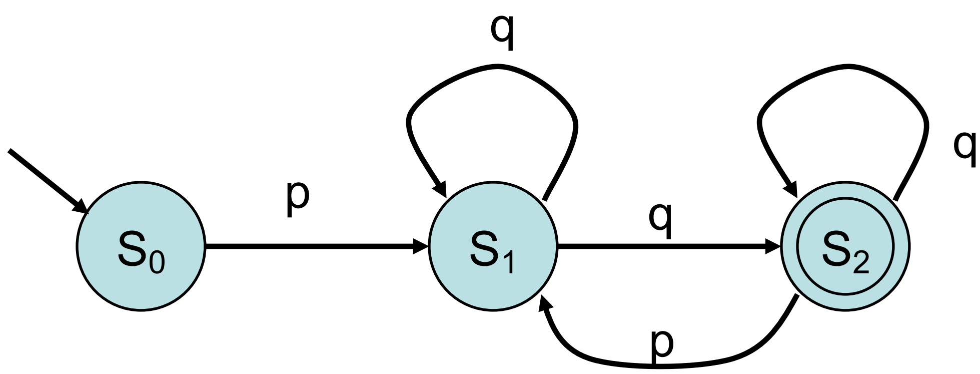
What words are accepted by this automaton B?
$L(B) = pq^+(pq^+)^*$

L(B) is called the language of B. It is the set of words for which there exists an accepting run of the automaton.

Non-determinism

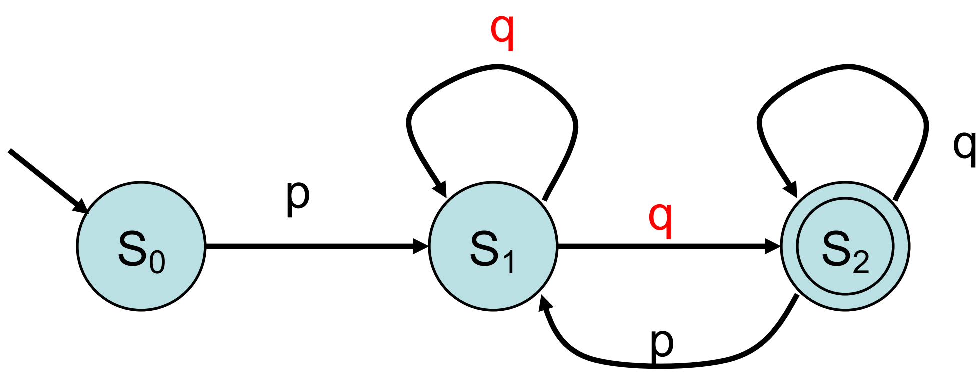
- Büchi automata are non-deterministic:
  - The next state is not uniquely defined
  - That is, the same input letter could lead to two different states

Example: Non-determinism



Example of non-determinism?

Example: Non-determinism



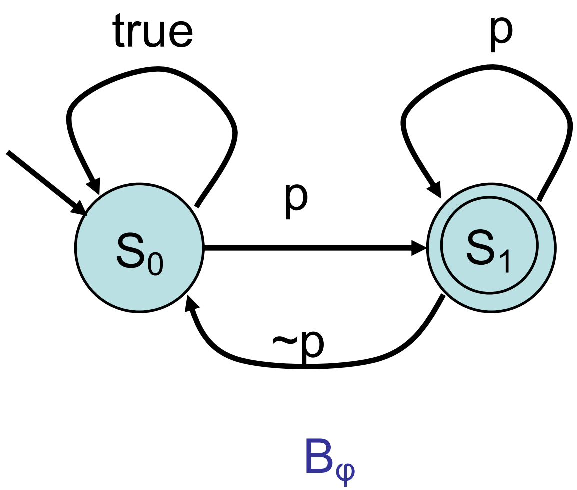
Non-determinism: (s_1,q,s_2) and (s_1,q, s_1) are in the transition relation δ

LTL to Buchi

- Every LTL formula has a corresponding *Buchi automaton* that accepts all and only the infinite state traces that satisfy the formula
- Example: $\varphi = G\ F\ p$

LTL to Buchi

- Every LTL formula has a corresponding *Buchi automaton* that accepts all and only the infinite state traces that satisfy the formula
- Example: φ = G F p

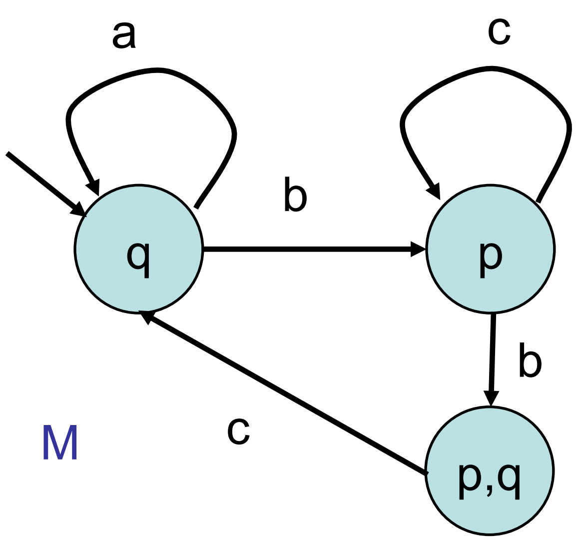


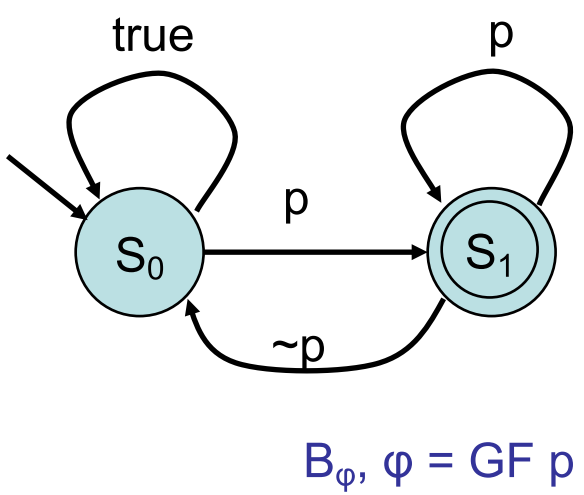
Checkpoint

- Where are we in the story?
  - What are we trying to do?
  - What are the pieces we assembled so far?

LTL Model checking

- TS M: input set A = {a,b,c} and AP={p,q}
- Formula φ = G F p
- Traces of M = infinite label sequences (e.g. σ_1=({q},{p},{p,q})* and σ_2={q}*)

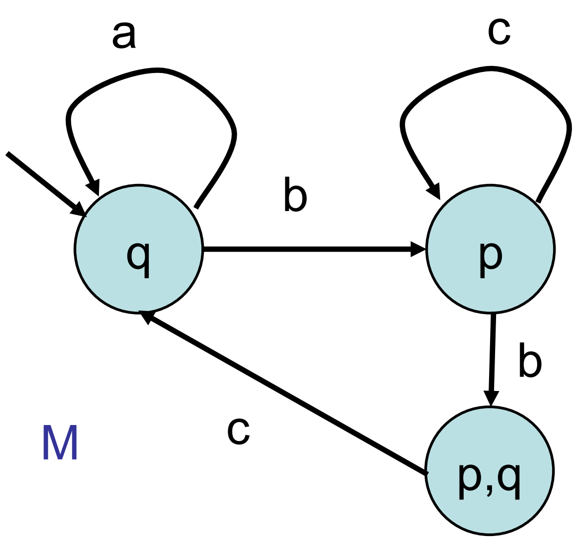


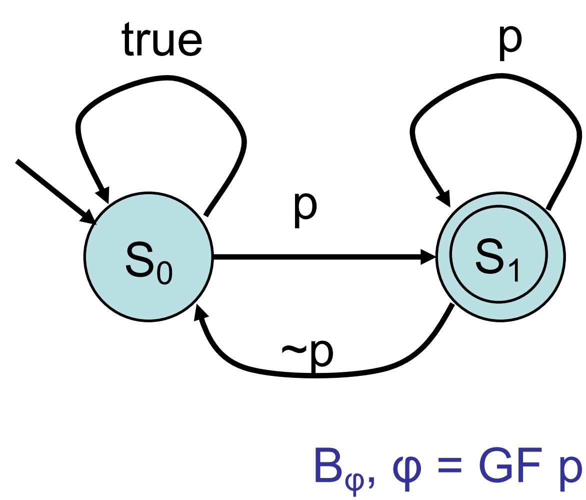


STUDENT

LTL Model checking

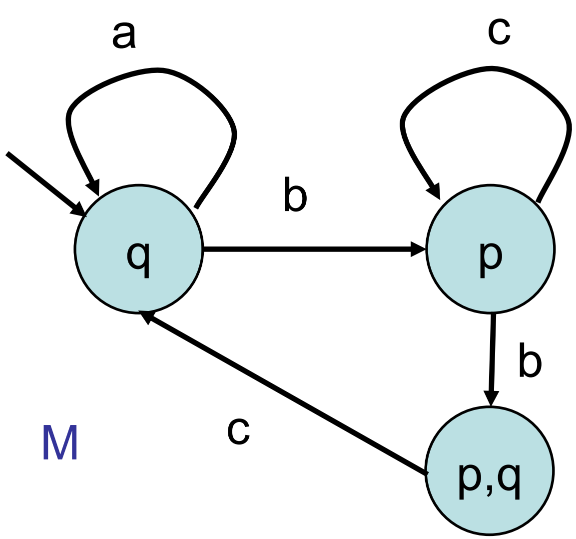
- TS M: input set A = {a,b,c} and AP={p,q}
- Not every trace of M satisfies formula. Give a counter-example

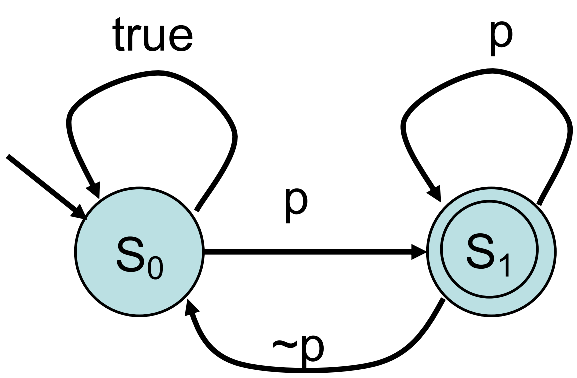




LTL Model checking

- TS M: input set A = {a,b,c} and AP={p,q}
- Not every trace of M satisfies formula. Counter-examples: σ_2={q}* and σ_3=qp{p,q}q*





B_φ, φ = GF p

LTL Model checking

- B_φ accepts exactly those traces that satisfy φ
- $B_{\sim\varphi}$ accepts exactly those traces that falsify (i.e., violate) φ
- Example (cont'd) :

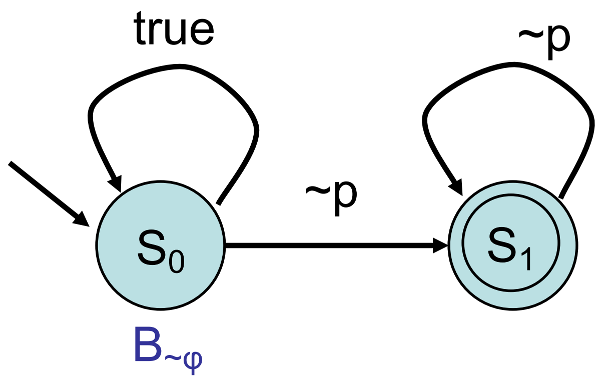
$$\sim\varphi = \sim(GFp) = F\sim(Fp) = F(G\sim p)$$

- What is $B_{\sim\varphi}$?

STUDENT

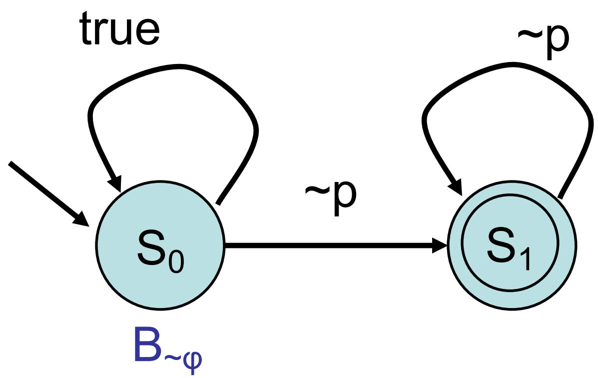
LTL Model checking

- B_φ accepts exactly those traces that satisfy φ
- $B_{\sim\varphi}$ accepts exactly those traces that falsify φ
- $\sim\varphi = \sim(GFp)=F\sim(Fp)=F(G\sim p)$



LTL Model checking

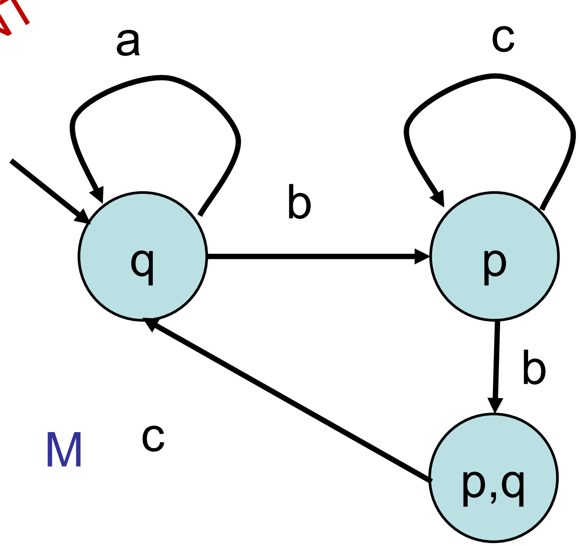
- If TS generates a trace that is accepted by $B_{\sim\varphi}$, this means, by construction, that the trace violates φ, and so that the TS is incorrect (relative to φ)

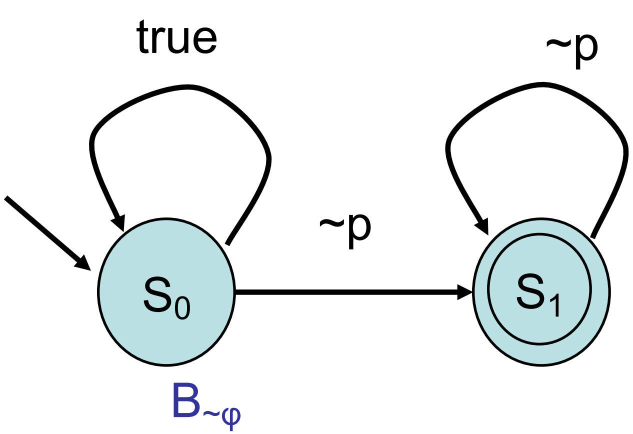


LTL Model Checking

- A trace of TS that is accepted by $B_{\sim\varphi}$ violates φ: TS is incorrect
- Imagine running the two automata in parallel: they both make transitions at the same time. If M transitions f → f' (f,f' in AP), $B_{\sim\varphi}$ transitions along the edges labeled by f'. $B_{\sim\varphi}$ <u>observes</u> M's operation.
- If every/no? such parallel execution is accepting in $B_{\sim\varphi}$, then M |= φ

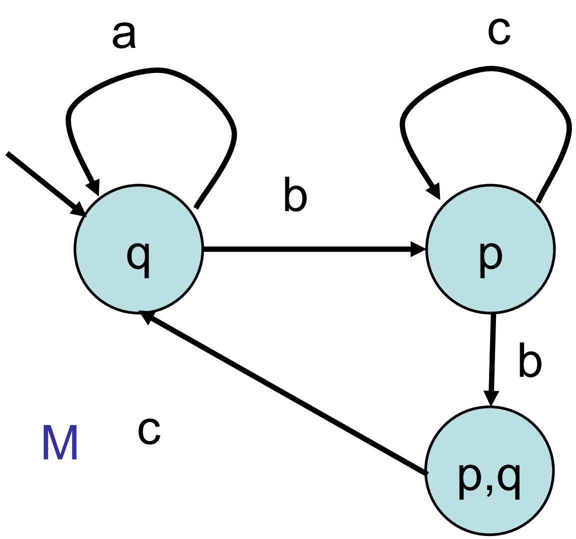
STUDENT



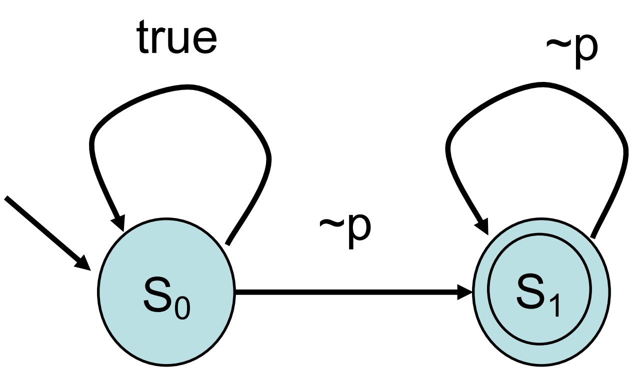


LTL Model Checking

- A trace of TS that is accepted by $B_{\sim\varphi}$ violates φ: TS is incorrect
- Imagine running the two automata in parallel: they both make transitions at the same time. If M transitions f → f' (f,f' in AP), $B_{\sim\varphi}$ transitions along the edges labeled by f'. $B_{\sim\varphi}$ observes M's operation.
- If **no** such parallel execution is accepting in $B_{\sim\varphi}$, then M $\models \varphi$



M

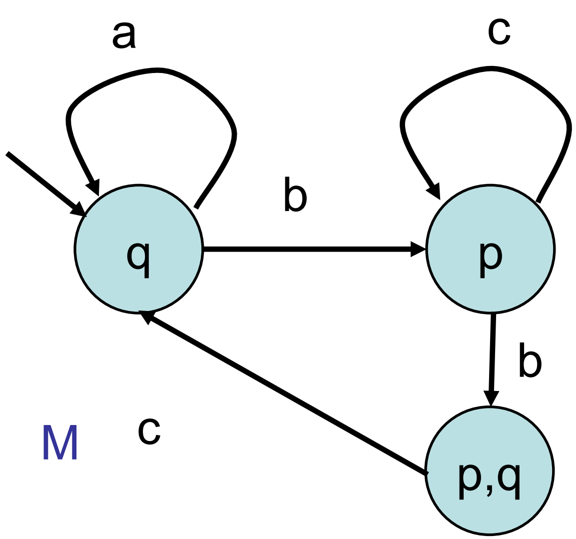


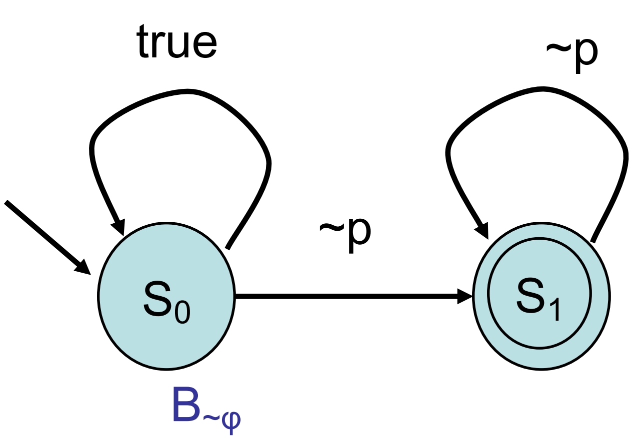
$B_{\sim\varphi}$

LTL Model Checking

- A trace of TS that is accepted by $B_{\sim\varphi}$ violates φ: TS is incorrect
- Find a counter-example (if any). I.e. a trace of M that is accepted by $B_{\sim\varphi}$

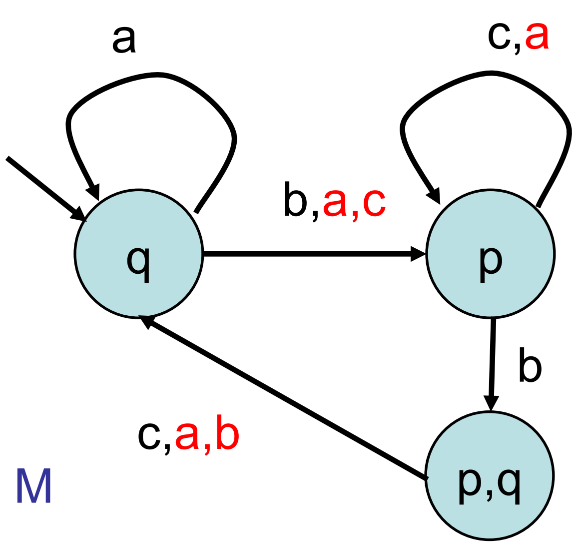
STUDENT

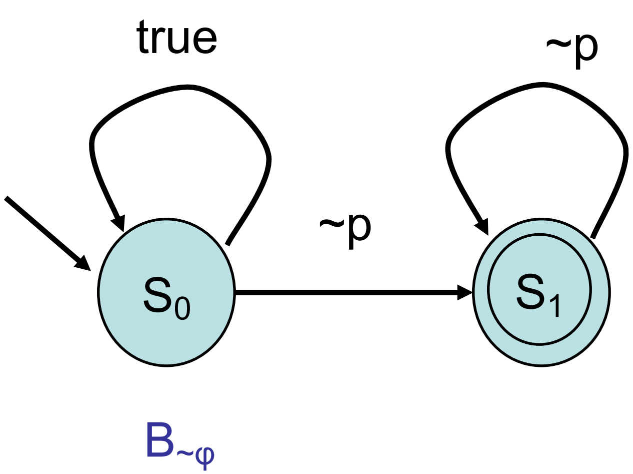




LTL Model Checking

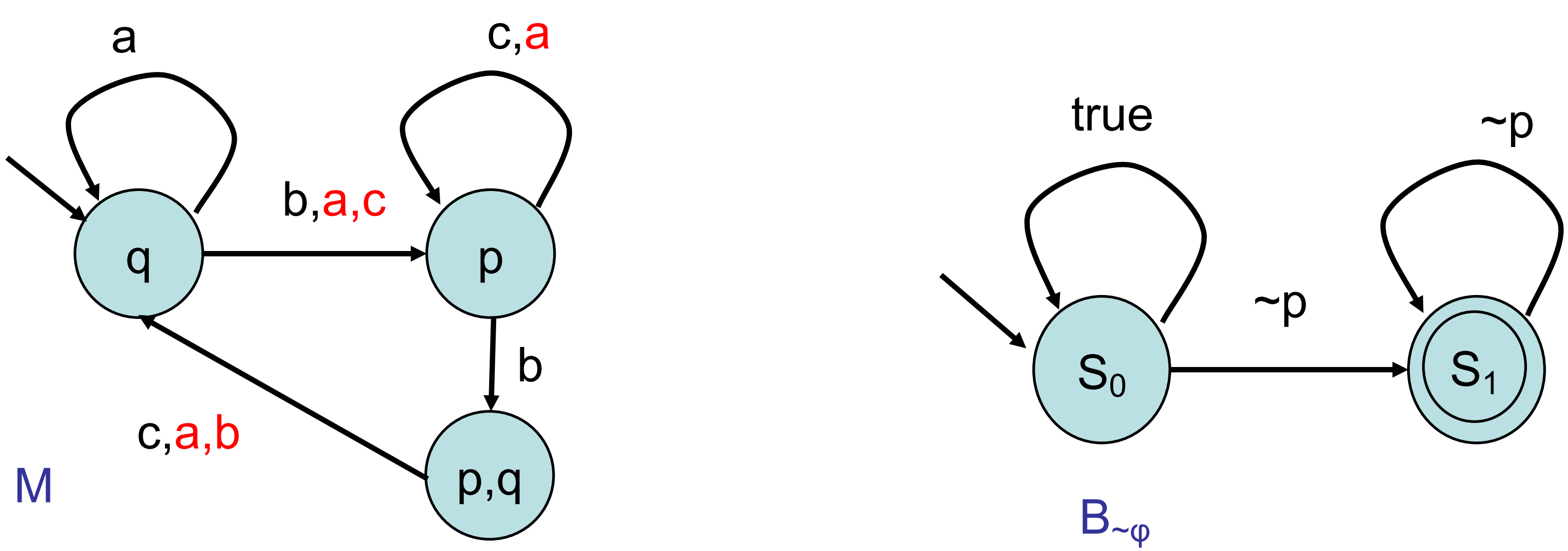
- A trace of TS that is accepted by $B_{\sim\varphi}$ violates φ: TS is incorrect
- Want to run the automata in parallel…





LTL Model Checking

- A trace of TS that is accepted by $B_{\sim\varphi}$ violates φ: TS is incorrect
- Want to run the automata in parallel…
- Take the product automaton!



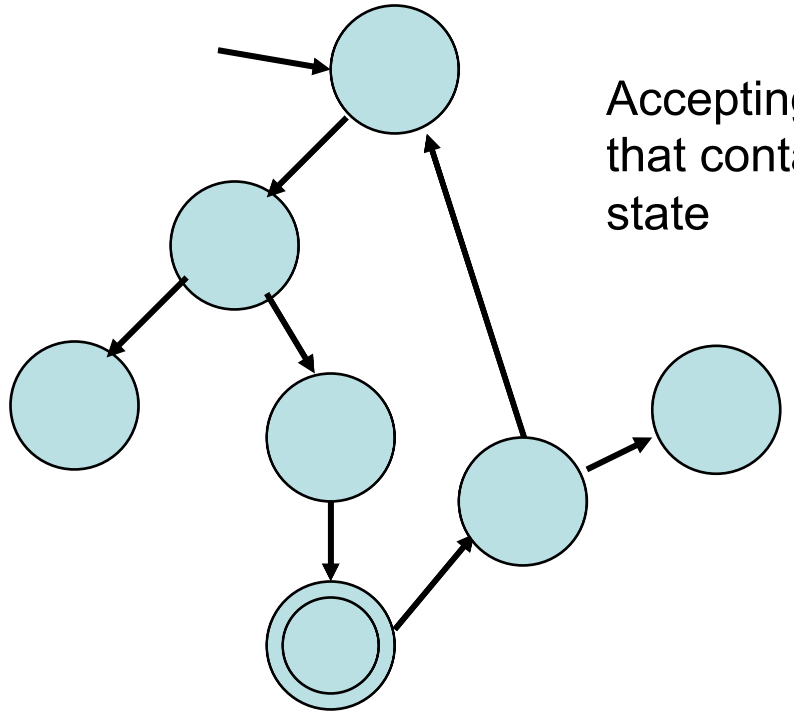
LTL Model Checking

- Given a model M and an LTL formula φ
  - Build the Buchi automaton $B_{\sim\varphi}$
  - Compute product of M and $B_{\sim\varphi}$
    - Each state of M is labeled with propositions
    - Each state of $B_{\sim\varphi}$ is labeled with propositions
    - Match states with the same labels
  - The product accepts the traces of M that are also traces of $B_{\sim\varphi}$ (i.e. $Tr(M) \cap L(\sim\varphi)$)
  - If the product accepts any sequence
    - We have found a counterexample

Nested Depth First Search

- The product is a Büchi automaton
- How do we find accepted sequences?
  - Accepted sequences must contain a cycle
    - In order to contain accepting states infinitely often
  - We are interested only in cycles that contain at least an accepting state
  - During depth first search start a second search when we are in an accepting states
    - If we can reach the same state again we have a cycle (and a **counterexample**)

Find an accepting trace

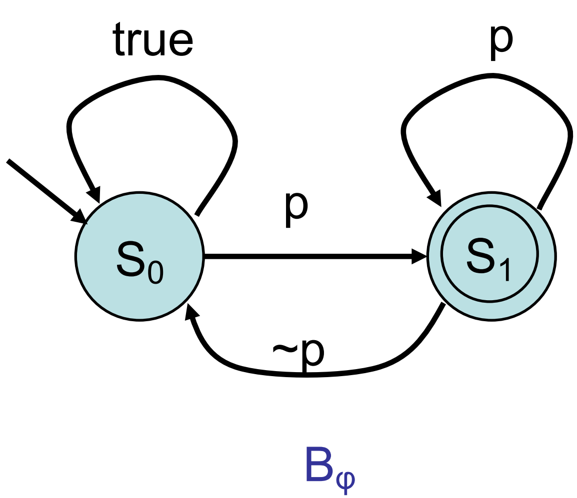


Accepting trace = cycle that contains an accepting state

Backup

LTL to Buchi complexity

- Every LTL formula of size n has a corresponding *Buchi automaton* of size $2^{O(n)}$ that accepts all and only the infinite state traces that satisfy the formula
- Example: G F p



Backup

LTL Model Checking

- Given a model M and an LTL formula φ
  - Check if All traces of M satisfy φ
  - $Tr(M) \subseteq S^\omega$ is the set of traces of M
  - $L(\varphi) \subseteq (2^{AP})^\omega$ is the language accepted by (the Buchi automaton of) φ
- M satisfies φ if $Tr(M) \subseteq L(\varphi)$
- Equivalently $Tr(M) \cap L(\sim\varphi) = \varnothing$